IBM® DB2® 地理情報エクステンダー

使用者の手引きおよび解説書

バージョン 7

SC88-8624-00
(英文原典：SC27-0701-00)

IBM® DB2® 地理情報エクステンダー

使用者の手引きおよび解説書

バージョン 7

SC88-8624-00
(英文原典：SC27-0701-00)

ご注意!

本書、および本書がサポートする製品をご使用になる前に、 321ページの『特記事項』にある一般的な情報を必ずお読みください。

本書において、日本では発表されていない IBM 製品 (機械およびプログラム)、プログラミング、またはサービスについて言及または説明する場合があります。しかし、このことは、弊社がこのような IBM 製品、プログラミング、またはサービスを、日本で発表する意図があることを必ずしも示すものではありません。

本マニュアルに関するご意見やご感想は、次の URL からお送りください。今後の参考にさせていただきます。

http://www.ibm.com/jp/manuals/main/mail.html

なお、日本 IBM 発行のマニュアルはインターネット経由でもご購入いただけます。詳しくは

http://www.ibm.com/jp/manuals/ の「ご注文について」をご覧ください。

(URL は、変更になる場合があります)

原 典:	SC27-0701-00 IBM® DB2® Spatial Extender User's Guide and Reference Version 7
発 行:	日本アイ・ビー・エム株式会社
担 当:	ナショナル・ランゲージ・サポート

第1刷 2000.6

この文書では、平成明朝体™W3、平成明朝体™W9、平成角ゴシック体™W3、平成角ゴシック体™W5、および平成角ゴシック体™W7を使用しています。この(書体*)は、(財) 日本規格協会と使用契約を締結し使用しているものです。フォントとして無断複製することは禁止されています。

注* 平成明朝体™W3、平成明朝体™W9、平成角ゴシック体™W3、
平成角ゴシック体™W5、平成角ゴシック体™W7

目次

図

表

本書について

本書は 2 部構成になっています。第 1 部では、DB2 地理情報エクステンダーの概念に関する情報が記載され、 Windows NT システムと AIX システムで DB2 地理情報エクステンダーのインストール、構成、管理、およびプログラミングを行う方法について説明されています。第 2 部では、DB2 地理情報エクステンダーで使用するストアード・プロシージャー、形状表現、関数、メッセージ、およびカタログ視点に関する参照情報が記載されています。

本書の対象読者

本書は、地理情報環境を設定する管理者と、地理情報データを扱うアプリケーションを開発するアプリケーション・プログラマーを対象としています。

表記規則

本書で使用されている、強調表示に関する規則は以下のとおりです。

太文字 (Boldface)
コマンドおよびグラフィカル・ユーザー・インターフェース (GUI) コントロールを示します (フィールドの名前、フォルダーの名前、メニュー選択項目など)。

モノスペース文字 (Monospace)
コーディングまたは入力テキストの例を示します。

イタリック体 (Italic)
値に置き換える必要のある変数を示します。また、資料の表題や強調語も示します。

英大文字 (UPPERCASE)
SQL キーワードおよびオブジェクトの名前 (表、視点、サーバーなど) を示します。

第1部 DB2 地理情報エクステンダーの使用

第1章 DB2 地理情報エクステンダーについて

この章では、DB2 地理情報エクステンダーの目的、処理されるデータ、および使用方法について説明します。この章の末尾には、残りの章へのクィック・ガイドが記載されています。

DB2 地理情報エクステンダーの目的

DB2 地理情報エクステンダーを使用して、*地理情報*システム (GIS) を作成します。 GIS とはオブジェクト、データ、アプリケーションの複合体で、これを使用すると地形に関する情報の生成や分析を行えます。*地形* には、地表面を形成しているオブジェクトや、地表面上にあるオブジェクトが含まれます。これらのオブジェクトは、自然環境 (河川、森、丘、砂漠など) と人工的な環境 (街、居住地、オフィスビル、陸標など) の両方から成ります。

地理情報 には以下の要素が含まれます。

- 環境面から見た地形の場所 (たとえば、ある街の中の病院や診療所がある場所を指したり、局地的な地震が起きたおおよその地域に含まれる街の所在地を指したりする)。
- 地形同士の関係 (たとえば、特定の地域内を流れる特定の水系に関する情報や、その地域内のその水系の支流に架かっている橋に関する情報など)。
- 1 つまたは複数の地形に適用される測定値 (たとえば、オフィスビルからその敷地の境界までの間の距離や、ある鳥が生息する境界線の長さ)

地理情報を単独で使用したり、従来のリレーショナル・データベース管理システム (RDBMS) の出力と一緒に使用したりすると、プロジェクトを設計したり、業務上の決定や戦略上の決定を下したりする際に役立ちます。たとえば、自治体の福祉管理者が、福祉援助の申請者と受取人が実際に住んでいる行政区内の地域を検査する必要があるとします。 DB2 地理情報エクステンダーを使用すると、行政区内の地域の位置と、申請者と受取人の住所からこの情報が導き出されます。

次に、レストラン・チェーンのオーナーが、近隣の街に事業を展開したいと思っているとします。新しい店を開く場所を決定するには、「これらの街の中のどの場所ならば、自分の店を頻繁に利用する常連客が集まるだろうか ? 主要な高速道路はどこにあるか ? 犯罪発生率が低いのはどこか ? 競争相手のレストランがある場所はどこか ?」という質問の答えを出す必要があります。 DB2

地理情報エクステンダーを使用するとこれらの質問の答えを求めて表示装置に地理情報を表示でき、基礎的な RDBMS を使用すると表示内容のラベルと説明テキストを生成できます。

本書には他にも DB2 地理情報エクステンダーの使用例がいくつか記載されています。特に、57ページの『第7章 地理情報の検索と分析』、61ページの『第8章 DB2 地理情報エクステンダー用のアプリケーションの作成』、および 169ページの『第14章 SQL 照会のための地理情報関数』を参照してください。

地形を表すデータ

以下に、地理情報を取得する際に生成し、格納し、操作するデータの概要について説明します。以下のトピックを取り上げます。

- データによって地形を表す方法
- 地理情報データの性質
- 地理情報データの作成方法

データによって地形を表す方法

DB2 地理情報エクステンダーでは地形を、表または視点の中の 1 行や行の一部分として表します。一例として、3ページの『DB2 地理情報エクステンダーの目的』 に挙げられている 2 つの地形 (オフィスビルと居住地) について考慮します。 図1 で、BRANCHES 表の個々の行は銀行の支店を表します。派生物として、図1 の CUSTOMERS 表の個々の行は、銀行の顧客を表します。しかし、個々の行の部分 (特に顧客の住所があるセル) が、顧客の居住地を表しているとみなすことができます。

BRANCHES

ID	NAME	ADDRESS	CITY	STATE	ZIP
937	Airzone-Multern	92467 Airzone Blvd	San Jose	CA	95141

CUSTOMERS

ID	LAST NAME	FIRST NAME	ADDRESS	CITY	STATE	ZIP	CHECKING	SAVINGS
59-6396	Kriner	Endela	9 Concourt Circle	San Jose	CA	95141	A	A

図 1. 表の行は地形を表す。表の行の住所データも地形を表す. BRANCHES 表のデータの行は銀行の支店を表している。 CUSTOMERS 表の住所データのセルは、顧客の居住地を表している。 2 つの表の名前と住所は架空のもの。

4ページの図1 の表には、銀行の支店と顧客を識別し、記述したデータがあります。この種のデータのことを、*属性データ* といいます。

属性データのサブセット (支店の住所と顧客の住所を示す値) を、地理情報を生成する値に変換できます。たとえば、4ページの図1 では、支店の住所が 92467 Airzone Blvd., San Jose CA 95141 になっています。顧客の住所は 9 Concourt Circle, San Jose CA 95141 です。 DB2 地理情報エクステンダーによってこれらの住所を、支店と顧客の家のある環境を示す値に変換できます。 図 2 は、 BRANCHES 表と CUSTOMERS 表の列にこのような値が入るよう指定して新しくした状態を示しています。

BRANCHES

ID	NAME	ADDRESS	CITY	STATE	ZIP	LOCATION
937	Airzone-Multern	92467 Airzone Blvd	San Jose	CA	95141	

CUSTOMERS

ID	LAST NAME	FIRST NAME	ADDRESS	CITY	STATE	ZIP	LOCATION	CHECKING	SAVINGS
59-6396	Kriner	Endela	9 Concourt Circle	San Jose	CA	95141		A	A

図 2. 地理情報列が追加された表. 個々の表の LOCATION 列には、住所に対応した座標が入れられます。

地理情報の開始点として住所および前述と同様の識別子を使用する場合、それらのデータのことをソース・データ といいます。ソース・データから導き出された値によって地理情報が生成されるので、導出値のことを*地理情報データ* といいます。次に、地理情報データについて説明し、関連するデータ・タイプを紹介します。

地理情報データの性質

多くの地理情報データは座標によって構成されます。*座標* とは、参照点からの相対的な位置を示す数値です。たとえば、緯度は赤道と比較した位置を示す座標です。また経度はグリニッジ子午線と比較した位置を示す座標です。したがって、イエローストーン国立公園の位置は、緯度 (赤道から北緯 44.45 度) と経度 (グリニッジ子午線から西経 110.40 度) で表されます。

緯度、経度、その参照点、および他の関連したパラメーターは、まとめて*座標系* とみなされます。緯度と経度以外の値に基づく座標系もあります。これらの座標系には独自の位置の測定値、参照点、および追加の識別パラメーターがあります。

最も単純な地理情報データ項目は、単一の地形の位置を定義した 2 つの座標から成り立ちます。 (*データ項目* とは、リレーショナル表のセルに入っている 1 つまたは複数の値のことです。) これより複雑な地理情報データ項目は、道路や河川などの線状の通り道を定義した複数の座標から成り立ちます。さらに複雑な項目は、領域の境界線 (たとえば、一区画の土地や氾濫原などの外縁) を定義した座標から成り立ちます。これらの地理情報データ項目や、 DB2 地理情報エクステンダーでサポートされている他の種類の地理情報データ項目に関する詳細は、 133ページの『第13章 図形および関連する地理情報関数』を参照してください。

個々の地理情報データ項目は、地理情報データ・タイプのインスタンスです。場所を示す 2 つの座標のデータ・タイプは ST_Point です。線状の道を定義する座標のデータ・タイプは ST_LineString です。境界線を定義する座標のデータ・タイプは ST_Polygon です。これらのタイプと、他のタイプの地理情報データは、単一の階層に属する構造型です。階層の概要については、35ページの『地理情報データ・タイプについて』を参照してください。

地理情報データのソース

地理情報データを取得するには、以下のようにします。

- 属性データから導出する。
- 他の地理情報データから導出する。
- インポートする。

属性データをソース・データとして使用する

DB2 地理情報エクステンダーでは、地理情報データを住所などの属性データから導出できます (4ページの『データによって地形を表す方法』を参照)。このプロセスのことを、*ジオコーディング* といいます。関係する一連の事柄について考慮するために、 5ページの図2 を「ジオコーディング実行前の」ピクチャー、 7ページの図3 を「ジオコーディング実行後の」ピクチャーとしましょう。 5ページの図2 の BRANCHES 表と CUSTOMERS 表は、地理情報データ用に指定された列が両方とも空です。 DB2 地理情報エクステンダーにより、これらの表の中の住所がジオコーディングされて住所に対応した座標が求められ、その座標がこの列に入れられるとします。この結果が 7ページの図3 に示されています。

BRANCHES

ID	NAME	ADDRESS	CITY	STATE	ZIP	LOCATION
937	Airzone-Multern	92467 Airzone Blvd	San Jose	CA	95141	1653 3094

CUSTOMERS

ID	LAST NAME	FIRST NAME	ADDRESS	CITY	STATE	ZIP	LOCATION	CHECKING	SAVINGS
59-6396	Kriner	Endela	9 Concourt Circle	San Jose	CA	95141	953 1527	A	A

図3. ソース・データから導出された地理情報データを含む表. CUSTOMERS 表の LOCATION 列には、ジオコーダーにより ADDRESS、CITY、STATE、および ZIP 列の住所から導出された座標が入っている。同様に、BRANCHES 表の LOCATION 列には、ジオコーダーによりこの表の ADDRESS、CITY、STATE、および ZIP 列の住所から導出された座標が入っている。この例は架空のもので、ここに示されている座標はシミュレーションであり実在しない。

DB2 地理情報エクステンダーでは、*ジオコーダー* という機能を使用して、属性データを地理情報データに変換し、その地理情報データを表列に入れます。ジオコーダーについて詳しくは、43ページの『ジオコーディングについて』を参照してください。

他の地理情報データをソース・データとして使用する

地理情報データは、属性データからだけでなく、他の地理情報データから生成することもできます。たとえば、支店が BRANCHES 表に定義されている銀行が、個々の支店から半径 5 マイル内に住んでいる顧客の数を知りたいとします。この銀行がデータベースから情報を取得するには、その前に個々の支店から半径 5 マイルの地帯をデータベースに定義しなければなりません。この種の定義は、DB2 地理情報エクステンダーの関数 ST_Buffer を使用して作成できます。 ST_Buffer により、個々の支店の座標を入力として使用して、希望する地帯の境界線を確定する座標を生成できます。 8ページの図4 は、 ST_Buffer によって BRANCHES 表に情報が入れられた様子を示しています。

BRANCHES

ID	NAME	ADDRESS	CITY	STATE	ZIP	LOCATION	SALES_AREA
937	Airzone-Multern	92467 Airzone Blvd	San Jose	CA	95141	1653 3094	1002 2001, 1192 3564, 2502 3415, 1915 3394, 1002 2001

図 4. 既存の地理情報データから導出された新しい地理情報データを含む表. SALES_AREA 列内の座標は、ST_Buffer によって LOCATION 列内の座標から導出された。 SALES_AREA 列内の座標は、LOCATION 列内の座標と同じくシミュレーションであり、実在しない。

DB2 地理情報エクステンダーには、ST_Buffer 以外にも、既存の地理情報データから新しく地理情報データを導出する関数があります。 ST_Buffer と他の関数の説明については、159ページの『既存の図形から新しい図形を生成する関数』を参照してください。

地理情報データのインポート

地理情報データを取得するための 3 つ目の方法は、 DB2 地理情報エクステンダーでサポートされているいずれかの形式で形式設定されているファイルからインポートすることです。この種の形式の説明については、293ページの『第16章 地理情報データのファイル形式』を参照してください。この種のファイルには、通常地図に適用されるデータが入ります。たとえば、人口調査標準地域、氾濫原、震源断層などです。この種のデータと、作成した地理情報データを組み合わせて使用すると、利用可能な地理情報を増やすことができます。たとえば、公共事業を行う省庁が、ある住宅地がどんな災害に遭いやすいか判別する場合に、 ST_Buffer を使用してその住宅地を含む地帯を定義できます。その後に、氾濫原や断層に関するデータをインポートして、これらの危険域が定義した地帯と重なり合っているかどうか調べることができます。

DB2 地理情報エクステンダー GIS の作成および使用方法

DB2 地理情報エクステンダー GIS を作成するには、DB2 地理情報エクステンダーとその基礎となる DB2 RDBMS を結合した環境で、 DB2 地理情報エクステンダーをセットアップして GIS プロジェクトを開発します。これらのプロジェクトを実装して、GIS を使用します。つまり、提供用に設計されている地理情報と従来の情報を両方とも生成して分析します。この作業全体で複数の一連のタスクを実行します。以下に、これらのタスクを実行するためのインターフェースを紹介し、タスクの概要を示し、それらを例示するシナリオを示します。

DB2 地理情報エクステンダーおよび関連機能に対するインターフェース

ここでは、DB2 地理情報エクステンダー GIS を作成して (DB2 地理情報エクステンダー GIS 用のリソースの設定や地理情報データの取得など)、使用する (地形に関する情報の生成および分析) ためのインターフェースについて概観します。

DB2 地理情報エクステンダー GIS を作成するには、以下のようにします。

- DB2 コントロール・センターの、DB2 地理情報エクステンダーのウィンドウとメニュー選択項目を使用する。指示については、以下を参照してください。
  - 25ページの『第3章 リソースの設定』
  - 35ページの『第4章 地理情報列を定義し、レイヤーとして登録して、ジオコーダーを使用して保守できるようにする』
  - 43ページの『第5章 地理情報列へのデータの取り込み』
  - 55ページの『第6章 地理情報索引の作成』
- DB2 地理情報エクステンダーのストアード・プロシージャーを呼び出すアプリケーション・プログラムを実行する。この種のプログラムの開発に関する指針については、61ページの『第8章 DB2 地理情報エクステンダー用のアプリケーションの作成』を参照してください。
- コントロール・センターとアプリケーション・プログラムを両方とも使用する。たとえば、コントロール・センターを使用してデフォルトのジオコーダーを起動できます。加えて別のジオコーダーも使用したい場合は、まずアプリケーション・プログラムで db2gse.gse_register_gc ストアード・プロシージャーを起動して、 DB2 地理情報エクステンダーに登録しなければなりません。 (デフォルト以外のジオコーダーに関する情報は、43ページの『ジオコーディングについて』を参照してください。 db2gse.gse_register_gc ストアード・プロシージャーに関する情報は、 96ページの『db2gse.gse_register_gc』を参照してください。)
- 他のインターフェースと、コントロール・センターかアプリケーション・プログラムの一方または両方を組み合わせて使用する。たとえば、地理情報機能 (ジオコーダーなど) によって生成されるデータを保持するための表を作成するには、コマンド行プロセッサーかコントロール・センターのインターフェースを使用できます。

DB2 地理情報エクステンダー GIS を使用するには、以下のようにします。

- 地理情報ブラウザー (ArcExplorer など。これは Environmental Systems Research Institute (ESRI) の製品) を使用して、情報を図形的にレンダリングする。

- DB2 コントロール・センターまたはコマンド行プロセッサーから SQL 照会を明示的に実行依頼する。
- アプリケーション・プログラムから SQL 照会を実行依頼する。

DB2 地理情報エクステンダー GIS の作成および使用のために実行するタスク

ここでは、DB2 地理情報エクステンダー GIS の作成や使用の際に用いるタスクを概説します。 GIS を作成する際のタスクには、DB2 地理情報エクステンダーのセットアップと GIS プロジェクトの開発が含まれます。 GIS を使用する際のタスクとしては、そのプロジェクトの実装があります。この概説ではまず DB2 地理情報エクステンダーのセットアップを取り上げ、次に GIS プロジェクトの開発と実装について説明します。最後に、概説の説明と実際の場面とで、タスクがどのように異なるかを示します。

DB2 地理情報エクステンダーのセットアップ

DB2 地理情報エクステンダーをセットアップするには、以下のようにします。

1. 計画と準備を行います (どんな GIS プロジェクトを開発するかの判断、DB2 地理情報エクステンダー用にどのデータベースを使用可能にするかの判断、 DB2 地理情報エクステンダーを管理する人の選択、プロジェクトの開発など)。
2. DB2 地理情報エクステンダーをインストールします。
3. リソースを適所に配置して GIS プロジェクトをサポートします。以下はその一例です。

   DB2 地理情報エクステンダーに備えられているリソース
   これにはシステム・カタログ、地理情報データ・タイプ、地理情報機能 (デフォルトのジオコーダーを含む) などが含まれます。これらのリソースのセットアップ・タスクのことを、*地理情報操作用のデータベースの使用可能化* といいます。

   ユーザー、ベンダー、またはその両者が開発したジオコーダー
   デフォルトのジオコーダーでは、米国の住所が地理情報データに変換されます。米国以外の住所や他の種類の属性データを地理情報データに変換するジオコーダーを、貴社または他社で開発することができます。

DB2 地理情報エクステンダーのインストールに関する指示は、17ページの『第2章 DB2 地理情報エクステンダーのインストール』を参照してください。コントロール・センターを使用してリソースを適所に配置することに関する指示は、 25ページの『第3章 リソースの設定』を参照してください。この目的でアプリケーション・プログラムを使用することに関する指針は、 61ページの『第8章 DB2 地理情報エクステンダー用のアプリケーションの作成』を参照してく

ださい。 DB2 地理情報エクステンダーのセットアップに関係するすべての作業を図示したシナリオについては、 13ページの『地理情報データと従来のデータを統合したシステム』を参照してください。

GIS プロジェクトの開発と実装

GIS プロジェクトを開発して実装するには、以下のようにします。

1. 計画と準備を行います (プロジェクトの目標の設定、必要な表とデータの判断、使用する座標系の判断など)。
2. 使用する地理情報参照システムを判別します。通常、座標値には正の整数、負の整数、および小数が含まれます。しかし、DB2 地理情報エクステンダーでは座標値はすべて正整数の形式で格納しなければなりません。*地理情報参照システム* とは、特定の座標系中の負数と小数が正整数に変換され、DB2 地理情報エクステンダーで格納できるようにする方法を定義するパラメーターの集合です。地理情報列にどの座標系を使用するか判断した後、地理情報列にとって必要な変換を行う地理情報参照システムを指定する必要があります。既存の地理情報参照システムが要件を満たしている場合はそのシステムを使用できます。満たしていない場合は作成できます。
3. 地理情報データを入れる列を 1 つまたは複数定義し、DB2 地理情報エクステンダーに登録し、ジオコーダーを使用して自動的に保守できるようにします。

   地理情報列を登録することには、DB2 地理情報エクステンダーのカタログに記録することが含まれます。列を登録した時点から、その列は*レイヤー*と呼ばれます。理由は、その列から生成される情報によって、 GIS で作成される仮想の地理ランドスケープに層、つまりレイヤーが追加されるからです。列の登録後に、地理情報に関する操作をその列に対して実行できます。たとえば、その列にデータを取り込んだり、地理情報索引を定義したりできます。
4. 地理情報列にデータを取り込みます。
   - ジオコーダーを必要とするプロジェクトでは、ジオコーダーのパラメーターを設定します。それからジオコーダーを実行して、使用可能なすべてのソース・データが 1 度の操作でコーディングされ、その結果の座標がレイヤーにロードされるようにします。
   - 地理情報データをインポートする必要があるプロジェクトでは、データをインポートします。
5. 地理情報列にアクセスしやすくします。特に、DB2 で地理情報データに即時アクセスできるようにするための索引を定義し、相互に関係のあるデータ

を効果的に検索できるようにするための視点を定義します。この種の視点を定義した後には、その地理情報列をレイヤーとして登録する必要があります。

6. 地理情報と、関連した業務情報を生成して分析します。この作業には、地理情報列および関連した属性列の照会が含まれます。この照会の際に、広範な情報を戻す DB2 地理情報エクステンダー関数を組み込むことができます。たとえば、2 つの地形の間の最短距離や、地形の周囲の地域を定義した座標などを戻せます。このような座標を戻す関数 ST_Buffer に関する情報は、7ページの『他の地理情報データをソース・データとして使用する』および195ページの『ST_Buffer』を参照してください。地理情報関数を使用した照会の例については、 57ページの『第7章 地理情報の検索と分析』および169ページの『第14章 SQL 照会のための地理情報関数』を参照してください。

コントロール・センターを使用して、 GIS プロジェクトの開発に関係したタスクを実行することに関する指示は、以下を参照してください。

- 25ページの『第3章 リソースの設定』
- 35ページの『第4章 地理情報列を定義し、レイヤーとして登録して、ジオコーダーを使用して保守できるようにする』
- 43ページの『第5章 地理情報列へのデータの取り込み』
- 55ページの『第6章 地理情報索引の作成』

コントロール・センターを使用して GIS プロジェクトを実装することに関する指針は、 57ページの『第7章 地理情報の検索と分析』を参照してください。

アプリケーション・プログラムを使用して GIS プロジェクトの開発と実装を行うことに関する指針は、 61ページの『第8章 DB2 地理情報エクステンダー用のアプリケーションの作成』を参照してください。

すべての作業を図示したシナリオについては、14ページの『オフィスを設立し、保険料を調整するプロジェクト』を参照してください。

一連のタスクの変化の可能性

DB2 地理情報エクステンダー GIS の作成と使用の際に実行する一連のタスクの内容と順序は、要件と使用するインターフェースによって変わります。たとえば、地理情報データを入れる列を定義し、レイヤーとして登録し、ジオコーダーを使用して自動的に保守できるようにするタスクについて考慮します。コントロール・センターを使用する場合は、これらのタスクは 1 つのウィンドウ

で一緒に実行します。しかし、プログラムからストアード・プロシージャーを起動する場合は、これらのタスクは個々に実行するので、いつ行うかを自分の判断で決められます。

シナリオ: 保険会社が自社の GIS を更新する場合

以下に、前述の一連のタスクを図示したシナリオを示します。

Safe Harbor 不動産保険会社の情報システム環境には、 DB2 ユニバーサル・データベース・システムが組み込まれており、またそれとは別に GIS データベース管理システムも組み込まれています。照会時に 2 つのシステムのデータの組み合わせを検索できるように拡張されています。たとえば、DB2 表には歳入に関する情報が格納され、 GIS 表にはこの会社の支店の場所が格納されているとします。この場合、指定した額の歳入をあげた事務所の場所を検索できます。しかし、2 つのシステムを統合することはできない (たとえば、 DB2 の列と GIS の列を結合できない) ので、照会の最適化などの DB2 サービスを GIS に使用することはできません。この種の不利な点を解消するために、 Safe Harbor 社は DB2 地理情報エクステンダーを購入して新しく GIS 開発部門を立ち上げます。以下に、この部門が DB2 地理情報エクステンダーをセットアップして、最初のプロジェクトを成し遂げる様子を示します。

地理情報データと従来のデータを統合したシステム

Safe Harbor の GIS 開発部門は、以下のようにして DB2 地理情報エクステンダーをセットアップします。

1. DB2 地理情報エクステンダーを DB2 環境に組み込む準備をします。たとえば次のようにします。
   a. この部門の管理チームが、DB2 地理情報エクステンダーのインストールと実装を行う地理情報管理チームと、地理情報の生成と分析を行う分析チームを指名します。
   b. Safe Harbor 社の業務上の決定は主に顧客の要件に基づいて行われるので、管理チームは、顧客に関する情報のあるデータベースに DB2 地理情報エクステンダーをインストールすることを決定します。この情報の大部分は CUSTOMERS という表に格納されます。

      GIS 開発部門のメンバーが、選択したデータベースを *GIS データベース* と呼べば、そのデータベースを参照する際に便利です。しかし、このデータベースは GIS プロジェクト専用として予約されているわけではありません。従来のように、地理情報アプリケーション以外のアプリケーションで使用される可能性があります。
2. 地理情報管理チームが DB2 地理情報エクステンダーをインストールします。

3. 地理情報管理チームが、GIS プロジェクトに必要なリソースをセットアップします。
   - コントロール・センターを使用して、地理情報操作に GIS データベースを使用できるようにするリソースを提供します。この種のリソースには、DB2 地理情報エクステンダー・カタログ、地理情報データ・タイプ、地理情報関数などが含まれます。
   - Safe Harbor 社は業務をカナダに展開し始めているので、地理情報管理チームは、カナダの住所を地理情報データに変換するジオコーダーをカナダの取引先に請求し始めます。

オフィスを設立し、保険料を調整するプロジェクト

GIS 開発部門は以下のようにして、DB2 地理情報エクステンダーの下で最初の GIS プロジェクトを遂行します。

1. プロジェクトを開発する準備をします。たとえば、次のようにします。
   - 管理チームが、プロジェクトの目標を設定します。
     - 新しい支店を設立する場所を決定します。
     - 顧客の危険地域 (交通事故の発生率が高い地域、犯罪発生率が高い地域、氾濫原、震源断層など) との近接の度合いに応じて、保険料を調整します。
   - GIS プロジェクトは、米国の顧客とオフィスのことを考慮します。したがって、地理情報管理チームは以下のことを決定します。
     - 座標系を使用して、Safe Harbor 社が業務を行う米国内の区域を正確に定義します。
     - デフォルトのジオコーダーは、米国の住所をジオコーディングするよう設計されているので、これを使用します。
   - 地理情報管理チームは、プロジェクトの目標を満たすのに必要なデータと、このデータを入れる表を決定します。
2. 地理情報管理チームは、コントロール・センターを使用して、2 つの地理情報参照システムを作成します。一方は、オフィスの場所を定義した座標が、DB2 地理情報エクステンダーで格納できるデータ項目に変換される方法を決定します。他方は、顧客の居住地を定義した座標が、 DB2 地理情報エクステンダーで格納できるデータ項目に変換される方法を決定します。
3. 地理情報管理チームは、コントロール・センターを使用して、地理情報データを入れる列を定義し、レイヤーとして登録し、ジオコーダーを使用して自動的に保守できるようにします。

- LOCATION 列を CUSTOMERS 表に追加します。この表には顧客の住所がすでに入っています。デフォルトのジオコーダーにより、これらのデータが地理情報データに変換され、 LOCATION 列にロードされます。
- OFFICES 表を作成して、独立した GIS に現在格納されているデータを入れます。このデータには、Safe Harbor 社の支店の住所、これらの住所からジオコーダーによって導出された地理情報データ、個々のオフィスから半径 5 マイル以内の地帯を定義した地理情報データが含まれます。ジオコーダーによって生成されたデータは、LOCATION 列に入れられます。地帯を定義したデータは、SALES_AREA 列に入れられます。
- 2 つの LOCATION 列と SALES_AREA 列をレイヤーとして登録します。
- デフォルトのジオコーダーによって 2 つの LOCATION 列が自動的に保守されるようにします。

4. 地理情報管理チームは、CUSTOMER 表の LOCATION 列、OFFICES 表全体、および新しい HAZARD_ZONES 表にデータを取り込みます。
   - コントロール・センターを使用して、CUSTOMER 表の LOCATION 列にデータを取り込みます。
     a. 以下の条件を満たす場合だけ、住所の地理情報データを LOCATION 列に挿入するようジオコーダーに指示します。「住所が米国国勢調査局の記録と 100 パーセントの正確さで一致しなければならない。」(国勢調査局によって提供される住所のファイルは、DB2 地理情報エクステンダーに付属しています。ジオコーダーでソース・データの住所を地理情報データに変換できるようにするには、その前に、この住所とファイル内の対応データの突き合わせを試行しなければなりません。何パーセントの正確さで一致すれば地理情報を表に入れるかをユーザーが指定します。このパーセント比率のことを、*精度* といいます。)
     b. 表内のすべての住所を 1 度の操作でジオコーディングできるように、バッチ・モードでジオコーダーを実行します。残念ながら、ジオコーダーでは約 10 分の 1 の住所が拒否されてしまいます ！
     c. 拒否されたのは新しい住所で、国勢調査局の記録に完全に一致したものがないからに違いないと推測されます。この問題を解決するために、精度を 85 に下げます。
     d. 再度バッチ・モードでジオコーダーを実行します。住所が拒否される比率が、受け入れられるレベルになります。

- 別の GIS に備えられているユーティリティーを使用して、オフィス・データをファイルにロードします。続いてコントロール・センターを使用して、このデータをファイルから新しい OFFICES 表にインポートします。
- コントロール・センターを使用して、HAZARD ZONES 表を作成し、その地理情報列をレイヤーとして登録し、データをインポートします。データは地図の製作会社が用意したファイルにあります。

5. 地理情報管理チームは、コントロール・センターを使用して、新しいレイヤーにアクセスしやすくします。
   - レイヤーの索引を作成します。
   - CUSTOMERS 表と HAZARD ZONES 表の列を結合した視点を作成します。そして、その視点の地理情報列をレイヤーとして登録します。
6. 地理情報分析チームは、照会を実行して、元の目標 (新しい支店を設立する場所の決定と、顧客の危険地域との近接度に応じた保険料の調整) を果たすのに役立つ情報を取得します。

第2章 DB2 地理情報エクステンダーのインストール

この章には、DB2 地理情報エクステンダーのインストールに関する指示が記載されています。以下のトピックが説明されています。

- 『DB2 地理情報エクステンダーの構成』
- 18ページの『システム要件』
- 19ページの『DB2 地理情報エクステンダーのインストール』
- 21ページの『インストールの検査』
- 22ページの『インストール後の考慮事項』
- 23ページの『次に行うこと』

DB2 地理情報エクステンダーの構成

DB2 地理情報エクステンダー・システムは、DB2 ユニバーサル・データベース、DB2 地理情報エクステンダー、および地理情報ブラウザー (ArcExplorer など) から構成されます。通常、地理情報操作に使用できるデータベースは、サーバー上にあります。クライアント・アプリケーションを使用し、 DB2 地理情報エクステンダーのストアード・プロシージャーと地理情報照会を介して地理情報データにアクセスします。また、地理情報ブラウザーを使用して地理情報データを表示できます。

図5 には、DB2 地理情報エクステンダーの体系が図示されています。

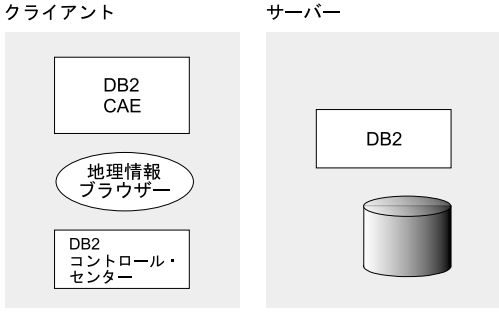


図5. クライアント / サーバーの設定

システム要件

以下に、DB2 地理情報エクステンダーのソフトウェアとハードウェアの要件について説明します。

サポートされているオペレーティング・システム

以下のオペレーティング・システム上で DB2 地理情報エクステンダーをインストールできます。

- AIX 4.2 またはそれ以降
- Windows NT 4.0 またはそれ以降、サービス パック 5 付き

必要なデータベース・ソフトウェア

DB2 地理情報エクステンダーをインストールする前に、ご使用のシステム上で DB2 ソフトウェアをインストールして構成しなければなりません。 表1 には、 DB2 地理情報エクステンダーのクライアント構成要素と DB2 地理情報エクステンダーのサーバー構成要素の両方のデータベース・ソフトウェア要件がリストされています。

表 1. 最小のソフトウェア要件

構成要素	ソフトウェア
クライアント	DB2 アドミニストレーション・クライアント バージョン 7.1[1]
サーバー	以下のいずれか。 • DB2 ユニバーサル・データベース エンタープライズ・エディション バージョン 7.1 • DB2 ユニバーサル・データベース エンタープライズ拡張エディション バージョン 7.1[2]
	注: 1. DB2 コントロール・センター、地理情報ブラウザー (地理情報データにアクセスするため)、または DB2 地理情報エクステンダーのサンプル・プログラムを使用する計画のない 場合は、古いバージョンの DB2 アドミニストレーション・クライアントを使用できます。 2. DB2 地理情報エクステンダーと DB2 ユニバーサル・データベース エンタープライズ拡張エディションは一緒に使用できますが、大量並列処理 (MPP) 環境の場合のように地理情報索引を複数のノードにまたがって分割することはできません。

ディスク・スペース要件

表2 には、推奨されている DB2 地理情報エクステンダーのディスク・スペース要件がリストされています。

表 2. ディスク・スペース要件

DB2 地理情報エクステンダー構成要素	ディスク・スペース
DB2 地理情報エクステンダーのサーバー・ライブラリー (DB2 地理情報エクステンダーのサーバー・ライブラリー、ジオコーダー参照データ、および資料を含む)	600 MB
DB2 地理情報エクステンダーのクライアント・サポート (サンプル・プログラム・データを含む)	15 MB

DB2 地理情報エクステンダーのインストール

以下に、 Windows NT および AIX オペレーティング・システム上で DB2 地理情報エクステンダーをインストールする際に必要な情報について説明します。

作業を始める前に

まだインストールしていない場合は、 DB2 アドミニストレーション・クライアント (コントロール・センターや実行時クライアントを含む管理ツール) をクライアント・ワークステーションにインストールし、 DB2 ユニバーサル・データベース エンタープライズ・エディションまたは DB2 ユニバーサル・データベース エンタープライズ拡張エディションをインストールしてください。これらのインストールに関する指示は、該当する*概説*および*インストール* を参照してください。

Windows NT システム上で DB2 地理情報エクステンダーをインストールする

Windows NT システム上で DB2 地理情報エクステンダーをインストールするには、以下のようにします。

1. 必要な管理許可を持つユーザー名でシステムにログオンします。
2. 他のプログラムをすべてシャットダウンします。
3. CD-ROM をドライブに挿入します。インストール・ランチパッドがオープンします。
4. 任意選択:**「リリース情報 (Release Notes)」**をクリックして、 DB2 地理情報エクステンダーのリリース情報を調べてインストール・プロセスに変更が加えられているか確認してから、 DB2 地理情報エクステンダーのランチパッドに戻ります。

5. **「インストール (Install)」**をクリックします。
6. セットアップ・プログラムのプロンプトに応答します。これ以降のステップを完了するための手引きとして、オンライン・ヘルプを使用できます。オンライン・ヘルプを起動するには、**「ヘルプ (Help)」**をクリックするか、F1 キーを押します。

インストールが完了すると、 DB2 地理情報エクステンダーは %DB2PATH% ディレクトリー (c:¥sqllib など) にインストールされます。

AIX システム上で DB2 地理情報エクステンダーをインストールする

AIX システム上で DB2 地理情報エクステンダーをインストールするには、以下のようにします。

1. root としてログインします。
2. CD-ROM をドライブに挿入します。
3. ご使用の AIX システムに CD-ROM をマウントします。 CD-ROM のマウントに関する情報は、*DB2 UDB (UNIX 版) 概説およびインストール* を参照してください。
4. 以下のコマンドを入力して、CD-ROM がマウントされたディレクトリーに変更します。

   ```
   cd /cdrom
   ```

   cdrom は、AIX 上の CD-ROM のマウント・ポイントです。
5. **db2setup** コマンドを入力して、 DB2 インストーラー・プログラムを開始します。「DB2 地理情報エクステンダーのインストール (Install DB2 Spatial Extender)」ウィンドウがオープンします。

   注: DB2 インストール・プログラムによりシステム上の情報がスキャンされるので、開始されるまで時間がかかります。
6. 「DB2 地理情報エクステンダーのインストール (Install DB2 Spatial Extender)」ウィンドウの製品リストから、インストールしたい製品を選択し、**「了解 (OK)」**をクリックします。

   DB2 地理情報エクステンダーをインストールする際の詳細や援助を得るには、**「ヘルプ (Help)」**をクリックします。

インストールが完了すると、 DB2 地理情報エクステンダーは /usr/lpp/db2_07_01 ディレクトリーにインストールされます。

インストールの検査

DB2 地理情報エクステンダーをインストールし終えたら、 DB2 地理情報エクステンダーのサンプル・プログラムを使用してインストール状況を検査できます。サンプル・プログラムを実行するには、その前に SAMPLE データベースを作成し、サンプル・プログラムを実行可能にしなければなりません。

注: DB2 地理情報エクステンダーの MAKE ファイルで指定されているコンパイラーを必ず使用してください。

Windows NT 版のサンプル・プログラムをコンパイルして実行するには、以下のようにします。

1. 管理者特権を持つユーザー ID でログインします。
2. コマンド行プロンプトで、 **db2sampl** を入力して、 DB2 SAMPLE データベースを作成します。
3. コマンド行プロンプトから、以下のコマンドを入力します。

   ```
   cd %DB2PATH%¥samples¥spatial
   ```

注: ステップ 3 を実行して、続けてインストールの検査を行うには、デフォルトの DB2 インスタンス (DB2-DB2) を所有している必要があります。

4. **make rungsedemo** と入力します。
5. **rungsedemo.exe** と入力します。
6. プログラムが実行されている間に表示されるエラー・メッセージと完了メッセージを検査します。

AIX 版のサンプル・プログラムをコンパイルして実行するには、以下のようにします。

1. root としてログインします。
2. DB2 インスタンスを作成または更新します。
3. コマンド行プロンプトで、 **db2sampl** を入力して、 DB2 SAMPLE データベースを作成します。
4. コマンド行プロンプトから、以下のコマンドを入力します。

   ```
   cd $DB2INSTANCE/sqllib/samples/spatial
   ```

注: ステップ 4 を実行して、続けてインストールの検査を行うには、作成または更新した DB2 インスタンスを所有している必要があります。

5. **make rungsedemo** と入力します。
6. **rungsedemo** と入力します。

7. プログラムが実行されている間に表示されるエラー・メッセージと完了メッセージを検査します。

サンプル・プログラムに関する情報については、61ページの『第8章 DB2 地理情報エクステンダー用のアプリケーションの作成』を参照してください。

インストール後の考慮事項

DB2 地理情報エクステンダーを正常にインストールし終えたら、以下の点を考慮する必要があります。

- ArcExplorer のダウンロード
- DB2 インスタンス更新ユーティリティーの実行

ArcExplorer のダウンロード

ArcExplorer Java 3.0 は、IBM 社がサンプル・プログラムとして配布していますが、 ESRI Web サイト http://www.esri.com から入手することもできます。

ArcExplorer のインストールと使用法について詳しくは、 *Using ArcExplorer* を参照してください。この資料も ESRI Web サイトで入手できます。

ArcExplorer には Java® 2 Runtime Environment (Standard Edition または Enterprise Edition) バージョン 1.2.2 が必要です。これは Sun Web サイト http://java.sun.com から無料で入手できます。

重要: DB2 ユニバーサル・データベース V7.1 には、IBM JDK 1.1.8 が付属しています。 ArcExplorer のために JRE 1.2.2 をインストールする際には、 DB2 とは別のディレクトリーにインストールしてください。 CLASSPATH 環境変数を適切に設定することを忘れないでください。

DB2 インスタンス更新ユーティリティー (db2iupdt) の実行

db2iupdt ユーティリティーは、指定された DB2 インスタンスを以下のように更新します。

- インスタンスを使用して新しいシステム構成を獲得できるようにする。
- インスタンスを使用して、特定の製品オプションのインストールや除去に関連した機能にアクセスできるようにする。

AIX では、このユーティリティーは /usr/lpp/db2_07_01 にあります。援助が必要な場合は、コマンド行で `db2iupdt -h` と入力してヘルプ・メニューをオープンします。 Windows NT オペレーティング・システムでは、 db2iupdt は

¥sqllib¥bin ディレクトリーにあります。このディレクトリーに変更して、上記のコマンドを入力してください。このコマンドの完全な説明については、*DB2 コマンド解説書* を参照してください。

次に行うこと

DB2 地理情報エクステンダーをインストールし終えたら、DB2 コントロール・センターを使用して GIS 環境を設定し、地理情報を処理し始めることができます。

DB2 地理情報エクステンダーをコントロール・センターから起動するには、以下のようにします。

1. 「コントロール・センター (Control Center)」ウィンドウから、オブジェクト・ツリーを展開して、DB2 地理情報エクステンダーの実行場所にしたいサーバーの下で**「データベース (Databases)」**フォルダーを見つけます。
2. **「データベース (Databases)」**フォルダーをクリックします。ウィンドウの右側の内容ペインに、データベースが表示されます。
3. 処理したいデータベースを右マウス・ボタン・クリックし、ポップアップ・メニュー内で実行したい地理情報操作をクリックします。

コントロール・センターからの DB2 地理情報エクステンダーの使用についての詳細は、以下を参照してください。

- 25ページの『第3章 リソースの設定』
- 35ページの『第4章 地理情報列を定義し、レイヤーとして登録して、ジオコーダーを使用して保守できるようにする』
- 43ページの『第5章 地理情報列へのデータの取り込み』
- 55ページの『第6章 地理情報索引の作成』

第3章 リソースの設定

DB2 地理情報エクステンダーをインストールし終えたら、地理情報列の作成時や地理情報データの操作時に必要なリソースをデータベースに備えます。この章では、この種のリソースについて要約し、これらのリソースを使用できるようにするための 2 つのタスク (データベースを地理情報操作に使用できるようにすることと、地理情報参照システムを作成すること) について説明します。

リソースの品目

地理情報列を作成する際や地理情報データを操作する際に利用するリソースには、以下のものが含まれます。

- *参照データ*: ジオコーディングしたい住所が DB2 地理情報エクステンダーで検査されるようにします。
- データベースを地理情報操作に使用できるようにするリソース: ストアード・プロシージャーや地理情報関数などです。
- ユーザーまたは他社製の、デフォルト以外のジオコーダー
- 地理情報参照システム

以下に、参照データと、データベースを地理情報操作に使用できるようにするリソースについて説明します。デフォルト以外のジオコーダーに関する情報は、 43ページの『ジオコーディングについて』を参照してください。地理情報参照システムに関する情報については、 27ページの『座標系および地理情報参照システムについて』を参照してください。

参照データ

参照データ は、米国の国勢調査局が集計した、米国内の最新の住所で構成されています。デフォルトのジオコーダーでデータベース内住所を座標に変換できるようにするには、その住所の一部またはすべてが参照データ内の住所と一致していなければなりません。

参照データは DB2 地理情報エクステンダーのインストール時に使用できるようになります。このデータに必要なディスク・スペースの量については、 19ページの『ディスク・スペース要件』を参照してください。 AIX 上でデータが適切にロードされたか検査するには、 $DB2INSTANCE/sqllib/gse/refdata/ ディ

レクトリーを探してください。 Windows NT 上でデータが適切にロードされたか検査するには、 %DB2PATH¥¥gse¥refdata¥ ディレクトリーを探してください。

データベースを地理情報操作に使用できるようにするリソース

DB2 地理情報エクステンダーのインストール後最初に実行するタスクとして、データベースを地理情報操作に使用できるようにします。これには、DB2 地理情報エクステンダーにより以下のリソースがデータベースにロードされるようにするアクションを開始することが含まれます。

- ストアード・プロシージャー。コントロール・センターからアクションを要求すると、 DB2 地理情報エクステンダーによりそのアクションを実行するストアード・プロシージャーの 1 つが起動されます。
- 地理情報データ・タイプ。地理情報データ・タイプを、地理情報データの格納先となる個々の表または視点に割り当てなければなりません。詳細については、35ページの『地理情報データ・タイプについて』を参照してください。
- DB2 地理情報エクステンダーのカタログ表または視点。 DB2 地理情報エクステンダーのカタログに従属している操作もあります。たとえば、地理情報データ・タイプのある列にデータを取り込めるようにするには、その前にカタログにレイヤーとして登録しなければなりません。レイヤーに関する情報については、11ページの『GIS プロジェクトの開発と実装』を参照してください。
- 地理情報索引タイプ。これを使用すると、レイヤーに関する索引を定義できます。
- 地理情報関数。これを使用して、さまざまな方法で地理情報データを処理できます。たとえば、地形間の関係を判別したり、地理情報データをさらに生成したりできます。この種の機能の 1 つとしてデフォルト・ジオコーダーがあります。これを使用すると、米国内の住所を座標に変換してから、その座標を地理情報列に挿入できます。地理情報関数について詳しくは、 133ページの『第13章 図形および関連する地理情報関数』および 169ページの『第14章 SQL 照会のための地理情報関数』を参照してください。デフォルト・ジオコーダーについて詳しくは、 43ページの『ジオコーディングについて』を参照してください。
- DB2GSE というスキーマ。これには上記に列挙したオブジェクトが含まれています。

コントロール・センターを使用して、これらのリソースのロードを開始する方法に関する指示は、 27ページの『データベースを地理情報操作に使用できるようにする』を参照してください。アプリケーション・プログラム内でルーチン

を使用して、同じタスクを実行することに関する指針については、 61ページの『第8章 DB2 地理情報エクステンダー用のアプリケーションの作成』を参照してください。

データベースを地理情報操作に使用できるようにする

地理情報操作にデータベースを使用できるようにするのに必要な許可について調べるには、 83ページの『許可』を参照してください。

コントロール・センターからデータベースを地理情報操作に使用できるようにするには、以下のようにします。

1. 「コントロール・センター (Control Center)」ウィンドウから、オブジェクト・ツリーを展開して、DB2 地理情報エクステンダーの実行場所にしたいサーバーの下で**「データベース (Databases)」**フォルダーを見つけます。
2. **「データベース (Databases)」**フォルダーをクリックします。ウィンドウの右側の内容ペインに、データベースが表示されます。
3. 希望するデータベースを右マウス・ボタン・クリックし、ポップアップ・メニューで**「地理情報エクステンダー (Spatial Extender)」** --> **「使用可能にする (Enable)」**をクリックします。地理情報列およびデータの作成と処理を行えるようにするためのリソースが、 DB2 地理情報エクステンダーによりデータベースに備えられます。

覚え書き: 地理情報操作にデータベースを使用できるようにするには、その前に、データベースのあるサーバーに DB2 地理情報エクステンダーをインストールしなければなりません。

地理情報参照システムの作成

以下に、地理情報参照システムと座標系の間の関係について説明し、コントロール・センターから地理情報参照システムを作成する方法について説明します。

座標系および地理情報参照システムについて

ここで、5ページの『地理情報データの性質』で始めた座標系の説明を続けます。次に、11ページの『GIS プロジェクトの開発と実装』 に記述されている地理情報参照システムの定義を拡張します。また、地理情報参照システムのパラメーターに割り当てる値を判別する際の指針についても説明します。

座標系、座標、および測定値

座標系とは、特定の地理区域に想像上の格子があるものとみなすことができます。例としては、地球を網羅する格子、ある国を網羅する格子、都道府県内の特定の領域を網羅する格子などがあります。区域内の個々の地形は、東西に走る標準線と南北に走る標準線の交差する箇所にあります。 *X 座標* という値は、東西の標準線上の位置を示します。 *Y 座標* というもう 1 つの値は、南北の標準線上の位置を示します。どちらの値も、格子の中央、つまり*原点* に対する位置を指します。

原点の X 座標 と Y 座標は共にゼロです。原点から東の方向の X 座標は正で、原点から西の方向の X 座標は負です。同様に、原点から北の方向の Y 座標は正で、原点から南の方向の Y 座標は負です。この分散図で、以下の一般例について考慮しましょう。座標系 A には、巨大都市の区域を網羅する格子があります。 X 座標 7 は、この格子の原点から測定値 7 単位分東の位置を示します。 X 座標 -9.5 は、この原点から測定値 9 つ半単位分西の位置を示します。

地理情報列中の個々のデータ項目には、 (1) 1 つの地形の場所を定義する 1 つの X 座標と 1 つの Y 座標、または (2) 1 つの地形の一部分の場所を定義するか、または 1 つの地形に網羅される区域を定義する複数の X 座標と複数の Y 座標があります。他の 2 種類の値 (*Z 座標* と*測定値*) も組み込めます。Z 座標と測定値は、X 座標や Y 座標とは違って、場所や区域を定義するために DB2 地理情報エクステンダーで使用されるのではありません。単に GIS アプリケーションで必要な情報を伝えるだけです。通常 Z 座標は、地形の高さや深さを示します。原点から上の方向の Z 座標は正で、原点から下の方向の Z 座標は負です。測定値は数値で、任意の種類の情報を伝えることができます。たとえば、GIS で油田を表そうとしているとします。地震データの震源地 ID を示す値を処理するアプリケーションが必要な場合は、これらの値を測定値として格納できます。

地理情報参照システム、オフセット、およびスケール因数

『座標系、座標、および測定値』で記述されているように、座標は負の数になることもあれば、小数で表されることもあります。同じことが測定値にも当てはまります。しかし、記憶域のオーバーヘッドを減らすために、 DB2 地理情報エクステンダーでは個々の座標と測定値が負ではない整数 (つまり正整数かゼロ) として格納されます。したがって、実際の負および小数の座標と測定値は、 DB2 地理情報エクステンダーで格納できるように負ではない整数に変換しなければなりません。また、変換方法を DB2 地理情報エクステンダーに指示する必要もあります。そのためには、特定のパラメーターを設定します。特

定の地理区域内の座標と測定値の変換に使用するパラメーター設定値のことを、まとめて*地理情報参照システム* といいます。

地理情報参照システムを作成するには、以下のようにします。

- 表そうとしている地形の、最小の負数の座標と測定値を判別する。 (ゼロとの差が大きい負の数ほど、値は小さくなります。 X 座標 -10 は X 座標 -5 より小さく、測定値 -100 は -50 より小さくなります。)
- *オフセット因数* (短く*オフセット* ともいう) を指定する。負の座標および測定値からこの値を減算して、負でない数字を求めます。
- *スケール因数* を指定する。小数の座標および測定値にこの値を乗算して、その座標および測定値と同じ精度か、それより高い精度の整数にします。たとえば、精度 4 の座標 92.77 があるとします。これにスケール因数 100 を乗算して、精度 4 の整数 9277 を求めることができます。

最小の負数の座標と測定値を判別する

地理情報参照システムのパラメーターを設定する前に、情報を得たい地形を含む地理区域内で最小の負数の X 座標、 Y 座標、Z 座標、および測定値を判別する必要があります。以下の質問に答えると、これらの値に該当するものを検出できます。

- 表そうとしている地形のうち、使用している座標系の原点から西方にあるものが存在するか ? 存在する場合、最も西にある地形の場所または西端を示す X 座標の値は何か ? (この答えが、扱っている範囲内で最小の負数の X 座標になります。) たとえば、油田を表そうとしていて、その一部が原点の西側にある場合、最も西側の油田の場所を示す X 座標は何か ?
- 原点から南方にある地形が存在するか ? 存在する場合、最も南にある地形の場所または南端を示す Y 座標の値は何か ? (この答えが、扱っている範囲内で最小の負数の Y 座標になります。) たとえば、油田を表そうとしていて、その一部が原点の南側にある場合、最も南側の油田の場所を示す Y 座標は何か ?
- Z 座標を使用して深さを定義しようとしている場合、最も深い地形はどれか、およびこの地形の最も深いポイントを表す Z 座標は何か ? (この答えが、扱っている範囲内で最小の負数の Z 座標になります。)
- 地理情報データに測定値を組み込もうとしている場合、負の数値のものがあるか ? ある場合、最小の負の測定値は何か ?

最小の負の座標と測定値を突き止めたら、その個々の値に、それぞれの値の 5 ～ 10 パーセントに相当する数値を加算してください。たとえば、最小の負の X 座標が -100 の場合、その値に -5 を加算できます。本書では、加算結果の数値を*増補値* と呼んでいます。

オフセット因数の指定

次に、DB2 地理情報エクステンダーで負の座標と測定値を、負でない数値に変換するのに使用するオフセット因数を指定します。

- 増補 X 値の値を判別したら、この値から減算するとゼロになる値をオフセットとして指定します。この指定後に、DB2 地理情報エクステンダーではすべての負の X 座標からこの数値を減算して、正の値にします。また、他のすべての X 座標からもこの数値を減算します。

  たとえば、増補 X 値が -105 の場合、計算結果を 0 にするにはこの値から -105 を減算する必要があります。すると DB2 地理情報エクステンダーにより、表そうとしている地形に関連したすべての X 座標から -105 が減算されます。これらの座標はすべて -100 以下であるため、減算結果の値はすべて正の数になります。
- 同様に、増補 Y 値、増補 Z 値、および増補測定値から減算するとゼロになる値をオフセットとして指定します。

X 座標から減算するオフセットのことを*偽の X* といいます。 Y 座標から減算するオフセットは*偽の Y*、 Z 座標から減算するオフセットは*偽の Z*、測定値から減算するオフセットは*偽の M* といいます。コントロール・センターからこれらのパラメーターを指定することに関する指示は、 31ページの『コントロール・センターから地理情報参照システムを作成する』を参照してください。

スケール因数の指定

次に、DB2 地理情報エクステンダーで小数の座標と測定値を整数に変換するのに使用するスケール因数を指定します。

- 小数の X 座標または Y 座標に乗算すると 32 ビットの整数になる値を、スケール因数として指定します。このスケール因数を 10 の累乗にすることをお勧めします。 10 の 1 乗 (10)、10 の 2 乗 (100)、10 の 3 乗 (1000) と求めていき、必要に応じてさらに大きい因数を求めます。スケール因数を因数 10 の何乗にする必要があるか判別するには、以下のようにします。
  1. 小数の、または小数と思われる X 座標と Y 座標を判別します。たとえば、多数の X 座標と Y 座標を扱っており、そのうちの 3 つが小数であるとします (1.23、5.1235、および 6.789)。
  2. 最も長精度の小数の座標に着目します。次に、この座標に 10 の何乗を因数として乗算すると、精度の等しい整数になるか判別します。現在の例では、3 つの小数の座標のうち、5.1235 が最も長精度です。 10 の 4 乗 (10000) を乗算すると、整数の 51235 になります。
  3. 上記の乗算によって求められる整数が、長すぎて 32 ビットのデータ項目として格納できないかどうかを判別します。 51235 は長すぎません。しかし、扱っている X 座標と Y 座標の中に、1.23、5.11235、および

6.789 以外にも、 4 つ目の小数として 10006.789876 があるとします。この座標の小数部の精度は他の 3 つの座標より長いので、この 座標 (5.1235 ではない) に 10 の累乗を乗算します。この値を整数に変換するには、10 の 6 乗 (1000000) を乗算することになります。しかし、計算結果の 10006789876 は、長すぎて 32 ビットのデータ項目として格納できません。 DB2 地理情報エクステンダーで格納が試行されると、結果は予測できないものになります。

この問題を避けるには、10 の累乗のうち、元の座標に乗算すると小数が求められるが、 DB2 地理情報エクステンダーで格納可能な整数に切り捨てる際に失われる精度が最小限になる値を選択します。この例の場合は、10 の 4 乗 (10000) を選択できます。 10006.789876 に 10000 を乗算すると、100067898.76 が求められます。 DB2 地理情報エクステンダーでこの数値は 100067898 に切り捨てられますが、正確でなくなる度合いは実質的には微々たるものです。

- 表そうとしている地形の Z 座標が小数の場合は、前述の手順に従って、それらの座標のスケール因数を確定します。またその地形に小数の測定値が関連付けられている場合は、同じ手順に従って、それらの測定値のスケール因数を確定します。

X 座標と Y 座標のスケール因数のことを、*XY 単位* といいます。 Z 座標のスケール因数のことを *Z 単位* といい、測定値のスケール因数のことを *M 単位* といいます。コントロール・センターからこれらのパラメーターを指定することに関する指示は、 『コントロール・センターから地理情報参照システムを作成する』を参照してください。

コントロール・センターから地理情報参照システムを作成する

ここでは、コントロール・センターから地理情報参照システムを作成するステップについて概説します。その概説に続いて、各ステップを完了する方法を詳しく説明します。

以下のステップを実行するのに、許可は必要ありません。

コントロール・センターから地理情報参照システムを作成するステップの概要:

1. 「地理情報参照の作成 (Create Spatial Reference)」ウィンドウをオープンします。
2. 使用したい座標系を指示します。
3. 作成したい地理情報参照システムの識別子を指定します。
4. 情報を得たい地形に適用される座標と測定値の範囲を判別します。

5. 負の数または小数の座標や測定値を、 DB2 地理情報エクステンダーで保管できるデータ項目に変換するのに使用できる値を指定します。
6. 必要な地理情報参照システムを作成するよう DB2 地理情報エクステンダーに指示します。

コントロール・センターから地理情報参照システムを作成するステップの詳細:

1. 「地理情報参照の作成 (Create Spatial Reference)」ウィンドウをオープンします。
   a. 「コントロール・センター (Control Center)」ウィンドウから、オブジェクト・ツリーを展開して、DB2 地理情報エクステンダーの実行場所にしたいサーバーの下で**「データベース (Databases)」**フォルダーを見つけます。
   b. **「データベース (Databases)」**フォルダーをクリックします。ウィンドウの右側の内容ペインに、データベースが表示されます。
   c. 地理情報データの操作に使用できるようにしたデータベースを右マウス・ボタン・クリックし、ポップアップ・メニューで**「地理情報エクステンダー (Spatial Extender)」** --> **「地理情報参照 (Spatial References)」**をクリックします。「地理情報参照 (Spatial References)」ウィンドウがオープンします。
   d. 「地理情報参照 (Spatial References)」ウィンドウから、**「作成 (Create)」**をクリックします。「地理情報参照の作成 (Create Spatial Reference)」ウィンドウがオープンします。
2. 「地理情報参照の作成 (Create Spatial Reference)」ウィンドウから、**「座標系 (Coordinate system)」**フィールドを使用して、使用したい座標系を指示します。
3. 作成したい地理情報参照システムの識別子を指定します。
   - **「名前 (Name)」**フィールドに、 1 ～ 64 文字のシステム名を入力します。

     制約事項: 別の地理情報参照システムの名前を指定しないでください。データベース内の 2 つの地理情報参照システムの名前を同じにすることはできません。
   - 「ID」フィールドで、数値識別子を入力します。この値は整数でなければなりません。

     制約事項: 別の地理情報参照システムの ID を指定しないでください。データベース内の 2 つの地理情報参照システムの ID を同じにすることはできません。

4. コントロール・センターの外部の媒体 (紙面やホワイト・ボードなど) を使用して、表そうとしている地形に適用される最小の負の座標と測定値を判別します。この方法に関する指針については、29ページの『最小の負数の座標と測定値を判別する』を参照してください。
5. 「地理情報参照の作成 (Create Spatial Reference)」ウィンドウから、負の数または小数の座標や測定値を、DB2 地理情報エクステンダーでサポートされるデータ項目 (つまり、 32 ビットの負でない整数) に変換する値を指定します。
   a. 負の数または小数の X 座標を、負でない整数に変換する値を指定します。
      - **「オフセット (Offset)」**列で、 **X** に最も近いフィールドに、偽の X を指定します。
        – 4 で識別した X 座標の範囲内の値に負の数がある場合は、偽の X として、最小の負の座標から減算すると正の数になる数値を入力します。指針については、30ページの『オフセット因数の指定』を参照してください。
        – X 座標がすべて負でない場合は、偽の X として 0 を入力します。
      - **「スケール因数 (Scale factor)」**列で、 **X** の右側遠方にあるフィールドで XY 単位を指定します。この XY 単位は、小数の X 座標か Y 座標に乗算すると、失う精度を最小限にしつつ 32 ビットのデータ項目として格納できる自然数が求められるような値にする必要があります。指針については、30ページの『スケール因数の指定』を参照してください。

        X の右側遠方にあるフィールドで XY 単位を指定し終えたら、 **Y** の右側遠方にあるフィールドにも同じ値が表示されます。
   b. DB2 地理情報エクステンダーで負の Y 座標を正の値に変換できるような値を、偽の Y として指定します。この操作は、**「オフセット (Offset)」**列の、 **Y** に最も近いフィールドで行います。
      - 4 で識別した Y 座標の範囲内の値に負の数がある場合は、偽の Y として、最小の負の座標から減算すると正の数になる数値を入力します。指針については、30ページの『オフセット因数の指定』を参照してください。
      - Y 座標がすべて正の場合は、偽の Y として 0 を入力します。
   c. 地理情報データに Z 座標を組み込もうとしている場合は、負の数または小数の Z 座標を、負でない整数に変換する値を指定します。

- **「オフセット (Offset)」**列で、 **Z** に最も近いフィールドに、偽の Z を入力します。
  - 33ページの4 で識別した Z 座標の範囲内の値に負の数がある場合は、偽の Z として、最小の負の座標から減算すると正の数になる数値を入力します。指針については、30ページの『オフセット因数の指定』を参照してください。
  - Z 座標がすべて負でない場合は、偽の Z として 0 を入力します。
- **「スケール因数 (Scale factor)」**列で、 **Z** の右側遠方にあるフィールドで Z 単位を指定します。この Z 単位は、小数の Z 座標に乗算すると、失う精度を最小限にしつつ 32 ビットのデータ項目として格納できる自然数が求められるような値にする必要があります。指針については、30ページの『スケール因数の指定』を参照してください。

d. 地理情報データに測定値を組み込もうとしている場合は、負の数または小数の測定値を正の整数に変換する値を指定します。
   - **「オフセット (Offset)」**列で、**「線形 (Linear)」**ラベルに最も近いフィールドに、偽の M を入力します。
     - 33ページの4 で識別した測定値の範囲内の値に負の数がある場合は、偽の M として、最小の負の測定値から減算すると正の数になる数値を入力します。指針については、30ページの『オフセット因数の指定』を参照してください。
     - 測定値がすべて正の場合は、偽の M として 0 を入力します。
   - **「スケール因数 (Scale factor)」**列で、**「線形 (Linear)」**ラベルの右側遠方にあるフィールドで M 単位を指定します。この M 単位は、小数の測定値に乗算すると、失う精度を最小限にしつつ 32 ビットのデータ項目として格納できる自然数が求められるような値にする必要があります。指針については、30ページの『スケール因数の指定』を参照してください。

6. **「了解 (OK)」**をクリックして、希望する地理情報参照システムを作成します。

第4章 地理情報列を定義し、レイヤーとして登録して、ジオコーダーを使用して保守できるようにする

DB2 地理情報エクステンダー GIS 用にリソースを設定し終えたら、地理情報データが入るオブジェクトを作成できます。たとえば、新しい表に地理情報データを入れる必要がある場合、地理情報データを入れたい列にそのデータのタイプを割り当てて、それらの表を定義できます。既存の表に地理情報列を追加する必要がある場合も、同じ操作を行えます。

新しい表や既存の表に地理情報列を入れる際には、この列をレイヤーとして登録する必要があります。また、ジオコーダーを使用して列にデータを取り込む計画がある場合は、列をレイヤーとして登録する際に、ジオコーダーでその列を自動的に保守できるようにすることができます。この作業を行うには、以下のようにします。DB2 地理情報エクステンダーでトリガーを定義し、地理情報列の対応する属性列 (複数可) が新規データや更新データを受け取るたびにジオコーダーが起動されるようにコード化します。ジオコーダーが起動されると、新しいデータや更新データが地理情報データに変換され、この地理情報データが地理情報列に入れられます。

表に地理情報列を定義した後で、この表列の上に視点列を作成することを選択できます。表列をレイヤーとして登録した後に、視点列をレイヤーとして登録しなければなりません。

この章では、地理情報列に割り当てることのできるデータ・タイプの性質と使用法について説明します。次に、コントロール・センターを使用して、表の地理情報列を定義し、この列をレイヤーとして登録し、ジオコーダーにより保守できるようにする方法を説明します。最後に、コントロール・センターを使用して、視点列をレイヤーとして登録する方法を説明します。

地理情報データ・タイプについて

以下に、地理情報列にとって必須のデータ・タイプについて説明し、地理情報列のデータ・タイプをどれにするか選択する際の指針を示します。

地理情報操作にデータベースを使用できるようにする際に、 DB2 地理情報エクステンダーによりデータベースに構造化データ・タイプの階層が作成されます。 36ページの図6 にこの階層が示されています。この図で、インスタンス化

できるタイプは背景が白くなっており、インスタンス化できないタイプは背景に陰影が付いています。

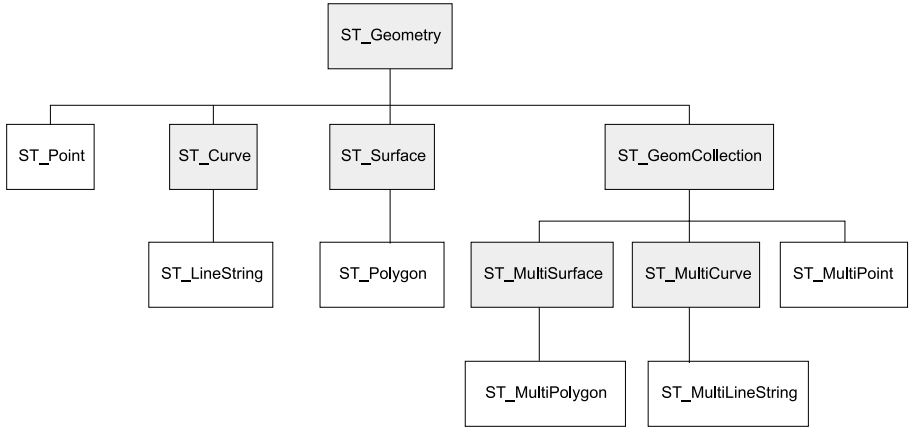


図 6. 地理情報データ・タイプの階層. 白いボックスに名前のあるデータ・タイプはインスタンス化できる。黒いボックスに名前のあるデータ・タイプはインスタンス化できない。

図6 中の階層には、以下のデータ・タイプが含まれています。

- 単一の単位から成っているとみなすことのできる地形のデータ・タイプ。たとえば、個人住居の敷地や孤立した湖など。
- 複数の単位または構成要素から成る地形のデータ・タイプ。たとえば、高速道路網や山脈など。
- すべての種類の地形のデータ・タイプ。

単一単位の地形データ・タイプ

ST_Point、ST_Linestring、および ST_Polygon を使用して、単一の単位から成るとみなすことのできる地形によって占められるスペースを定義した座標を格納します。

- ST_Point を使用して、孤立した地形によって占められる地点を指示します。この種の地形は非常に小さい場合 (井戸など) や、非常に大きい場合 (街など) や、その中間の大きさの場合 (団地や公園など) があります。いずれの場合でも、スペースを示す地点は、東西に走る座標軸 (緯線など) と南北に走る座標軸 (経線など) の交点にすることができます。 ST_Point データ項目には、この交点を定義した値 (X 座標と Y 座標) が含まれます。 X 座標は東西に走る線上の交点を示し、Y 座標は南北に走る線上の交点を示します。
- ST_Linestring は、線形の地形 (街路、水路、配管など) によって占められるスペースを定義する座標用に使用します。

- ST_Polygon は、多面体の地形 (福祉行政区、森、野生生物の生息地など) によって網羅されるスペースの範囲を示したい場合に使用します。 ST_Polygon データ項目は、この種の地形の境界線を定義した座標群から成ります。

  ST_Polygon と ST_Point を同じ地形に使用できる場合があります。たとえば、団地群に関する地理情報が必要であるとします。個々の団地があるスペースの地点を表したい場合は、ST_Point を使用して、個々の地点を定義した X 座標と Y 座標を格納します。他方、個々の団地によって網羅されている区域を表したい場合は、 ST_Polygon を使用して、個々の区域の境界線を定義します。

複数の単位から成る地形のデータ・タイプ

ST_MultiPoint、ST_MultiLineString、および ST_MultiPolygon を使用して、複数の単位から成る地形によって占められるスペースを定義した座標を格納します。

- ST_MultiPoint を使用して、離散的な単位から成る地形を表したり、個々の構成要素によって占められるスペースの地点を指示します。 ST_MultiPoint データ項目には、この種の地形の個々の構成要素の場所を定義した X 座標と Y 座標の複数の対が含まれます。たとえば、ある表の行が群島を表し、列の中に ST_MultiPoint 列が含まれているとします。この列の個々のデータ項目には、個々の群島の場所を定義した X 座標 Y 座標の複数の対が含まれます。
- ST_MultiLineString を使用して、線形の複数の単位から成る地形を表し、個々の単位によって占められるスペースに関する情報を指示します。 ST_MultiLineString データ項目は、この種のスペースを定義した座標群から成ります。たとえば、ある表の行が水系を表し、列の中に ST_MultiLineString 列が含まれているとします。この列の個々のデータ項目には、個々の水系中の河川の流域を定義した X 座標 Y 座標の複数の対が含まれます。
- ST_MultiPolygon を使用して、多面体の複数の単位から成る地形を表し、個々の単位によって占められるスペースに関する情報を指示します。たとえば、ある表の行が中西部の郡を表し、列の中に ST_MultiPolygon 列が含まれているとします。この列には、農地に関する情報が含まれます。特に、この列の個々のデータ項目には、特定の郡の農地の境界線を定義した座標の集合が含まれます。

すべての地形のデータ・タイプ

どのデータ・タイプを使用するか分からない場合は、ST_Geometry を使用できます。 ST_Geometry は、他のデータ・タイプが属する階層のルートなので、他のデータ・タイプに割り当てられた列に格納できる値は、すべて ST_Geometry 列に格納できます。

重要: デフォルトのジオコーダーにより、住所を ST_Point または ST_Geometry データ項目に変換できます。したがって、このジオコーダーを使用して地理情報列にデータを取り込む計画がある場合は、この列に ST_Point または ST_Geometry データ・タイプを割り当てなければなりません。

表に地理情報列を定義し、この列をレイヤーとして登録し、ジオコーダーにより保守できるようにする

以下に、表に地理情報列を定義し、この列をレイヤーとして登録し、ジオコーダーにより保守できるようにするステップの概要を説明します。その概説に続いて、各ステップを完了する方法を詳しく説明します。

表列をレイヤーとして登録するのに必要な許可を調べるには、 98ページの『許可』を参照してください。ジオコーダーを使用してこの列を保守できるようにするのに必要な許可を調べるには、 79ページの『許可』を参照してください。

表に地理情報列を定義し、この列をレイヤーとして登録し、ジオコーダーにより保守できるようにするステップの概要:

1. 地理情報列を新しい表の一部にする場合は、この表を作成します。
2. 「地理情報レイヤーの作成 (Create Spatial Layer)」ウィンドウをオープンします。
3. 地理情報列を表に追加して、この列をレイヤーとして登録したいことを指示します。または、既存の列をレイヤーとして登録したいことを指示します。
4. レイヤーに使用する地理情報参照システムを指示します。
5. インポートされたデータや、別の地理情報列から生成されたデータをレイヤーに入れる場合は、レイヤーを作成するよう DB2 地理情報エクステンダーに指示します。
6. 属性データから導出されたデータをレイヤーに入れる場合は、以下のようにします。
   a. この属性データを入れる列を 1 つまたは複数指定します。
   b. ジオコーダーを使用してレイヤーを保守できるようにしたいことを指示します。

c. レイヤーを作成するよう DB2 地理情報エクステンダーに指示します。

表に地理情報列を定義し、この列をレイヤーとして登録し、ジオコーダーにより保守できるようにするステップの詳細:

1. 地理情報列を新しい表の一部にする場合は、この表を作成します。
   - 任意に選んだインターフェース (コントロール・センターやコマンド行プロセッサーなど) を使用して、表を作成します。
   - ジオコーダーを使用する計画がある場合は、ジオコーダーで操作される列を 1 ～ 10 個作成します。ジオコーダーでは 11 列以上のデータは入力として使用できません。
   - レイヤーとして登録する地理情報列を組み込むか、 3 のステップでこの列を定義します。

   既存の表を使用する場合は、次のステップに進みます。
2. 「地理情報レイヤーの作成 (Create Spatial Layer)」ウィンドウをオープンします。
   a. 「コントロール・センター (Control Center)」ウィンドウから、オブジェクト・ツリーを展開して、地理情報操作に使用したいデータベース内の表の**「表 (Tables)」**フォルダーを見つけます。
   b. **「表 (Tables)」**フォルダーをクリックします。ウィンドウの右側にある内容ペインに表が表示されます。
   c. 希望する表を右マウス・ボタン・クリックし、ポップアップ・メニューで**「地理情報エクステンダー (Spatial Extender)」** --> **「地理情報レイヤー (Spatial Layers)」**をクリックします。「地理情報レイヤー (Spatial Layers)」ウィンドウがオープンします。
   d. 「地理情報レイヤー (Spatial Layers)」ウィンドウから、**「作成 (Create)」**をクリックします。「地理情報レイヤーの作成 (Create Spatial Layer)」ウィンドウがオープンします。
3. 「地理情報レイヤーの作成 (Create Spatial Layer)」ウィンドウから、地理情報列を表に追加して、この列をレイヤーとして登録したいことを指示します。または、既存の列をレイヤーとして登録したいことを指示します。
   - 地理情報列を表に追加して、この列をレイヤーとして定義するには、以下のようにします。
     a. **「レイヤー列 (Layer column)」**フィールドに、列の名前を入力します。

b. **「列タイプ (Column type)」**フィールドで、列のデータ・タイプとして指定したいものを選択するか入力します。指定できるデータ・タイプに関する説明は、 35ページの『地理情報データ・タイプについて』を参照してください。

- 既存の列をレイヤーとして定義したい場合は、**「レイヤー列 (Layer column)」**フィールドでその列を選択します。

  制約事項: すでにレイヤーとして定義されている列を選択しないでください。

4. **「地理情報参照名 (Spatial reference name)」**で、レイヤーに使用する地理情報参照システムの名前を指定します。
5. インポートされたデータや、別の地理情報列から生成されたデータをレイヤーに入れたい場合は、**「了解 (OK)」**をクリックして登録します。
6. 属性データから導出されたデータをレイヤーに入れたい場合は、以下のようにします。
   a. この属性データを入れる列を 1 つまたは複数指定します。
      1) **「使用可能列 (Available columns)」**ボックスで、 1 つまたは複数の列を選択します。最大 10 列まで選択できます。
      2) **>** 押しボタン、 **>>** 押しボタン、またはその両方をクリックして、選択した列を**「選択列 (Selected columns)」**ボックスにリストします。
   b. ジオコーダーを使用してレイヤーを保守できるようにしたい場合は、以下のようにします。
      1) **「ジオコーダーを自動的に使用可能にする (Enable automatic geocoder)」**チェック・ボックスを選択します。
      2) **「名前 (Name)」**フィールドで、使用したいジオコーダーの名前を選択します。
      3) **「精度レベル (Precision level)」**フィールドで、入力レコードが処理されるには参照データ内の対応するレコードとどの程度一致しなければならないかを、パーセント単位で指定します。このパーセント比率のことを、*精度* といいます。たとえば、557 Bailey, San Jose 94120 という住所の入力レコードがジオコーダーに読み取られたとします。精度が 100 の場合には、この住所と参照データ内の対応レコードとが 100 パーセントの正確さで一致していなければ、この住所はジオコーダーに拒否されます。精度が 75 の場合には、この住所と参照データ内の対応レコードとが 75 パーセント以上の正確さで一致していれば、この住所はジオコーダーに処理されます。

4) 他社製のジオコーダーの場合は、**「特性 (Properties)」**ボックスを使用して、使用したい他社製のジオコーダーのパラメーターを指定します。

c. **「了解 (OK)」**をクリックして、選択した列をレイヤーとして登録し、要求に応じて、ジオコーダーを使用してその列を保守できるようにします。

視点列をレイヤーとして登録する

視点列をレイヤーとして登録するのに必要な許可を調べるには、 98ページの『許可』を参照してください。

視点列をレイヤーとして登録するには、以下のようにします。

1. 「地理情報レイヤーの作成 (Create Spatial Layer)」ウィンドウをオープンします。
   a. 「コントロール・センター (Control Center)」ウィンドウから、オブジェクト・ツリーを展開して、地理情報操作に使用したいデータベース内の視点の**「視点 (Views)」**フォルダーを見つけます。
   b. **「視点 (Views)」**フォルダーをクリックします。ウィンドウの右側の内容ペインに、視点が表示されます。
   c. 希望する視点を右マウス・ボタン・クリックし、ポップアップ・メニューで**「地理情報エクステンダー (Spatial Extender)」** --> **「地理情報レイヤー (Spatial Layers)」**をクリックします。「地理情報レイヤー (Spatial Layers)」ウィンドウがオープンします。
   d. 「地理情報レイヤー (Spatial Layers)」ウィンドウから、**「作成 (Create)」**をクリックします。「地理情報レイヤーの作成 (Create Spatial Layer)」ウィンドウがオープンします。
2. **「レイヤー列 (Layer column)」**ボックスを使用して、レイヤーとして登録したい列を指定します。
3. **「基礎となる地理情報レイヤー (Underlying spatial layer)」**フィールドで、選択した視点列が属している表列の名前を指定します。この表列はすでにレイヤーとして登録されていなければなりません。
4. **「了解 (OK)」**をクリックして、視点列をレイヤーとして登録します。

第5章 地理情報列へのデータの取り込み

地理情報列をレイヤーとして登録したら、各列に地理情報データを提供できます。 6ページの『地理情報データのソース』に記されているように、地理情報データを提供するには 3 つの方法、すなわち、属性データからデータを導出するジオコーダーと呼ばれる関数の使用、他の地理情報データからデータを導出するその他の関数の使用、そしてファイルからデータをインポートする方法があります。この章では、以下の点を説明します。

- ジオコーディングの解説、およびコントロール・センターを使って属性データをバッチ・モードでジオコーディングする方法
- データのインポートおよびエクスポート、およびコントロール・センターを使って GIS との間でデータをインポートおよびエクスポートする方法

既存の地理情報データから新しい地理情報データを導出するための関数についての詳細は、 159ページの『既存の図形から新しい図形を生成する関数』を参照してください。

ジオコーダーの使用

ここでは、ジオコーディングのプロセスを示し、コントロール・センターからジオコーダーをバッチ・モードで実行する方法について説明します。

ジオコーディングについて

この節では、ジオコーダーとそのソース間に見られる基本的な相違を区別します。また、ジオコーダーを操作できる 2 つのモードについて説明し、ジオコーダーの使用を計画する際に考慮すべき要素も取り上げます。

DB2 地理情報エクステンダーを使用すると、次のことを行えます。

- DB2 地理情報エクステンダーに付属するデフォルト・ジオコーダーの使用。
- 第三者ベンダーによって開発されたジオコーダーのプラグイン。
- 独自のジオコーダーのプラグイン。

デフォルト・ジオコーダーは米国の住所をジオコーディングし、 ST_Point データまたは ST_Geometry データのいずれかに変換できます。その他の地理情報データ・タイプのデータを格納する必要がある場合は、そのようなデータを生成するためのジオコーダーをプラグインできます。米国以外の場所、または

住所を持たない場所 (地目が異なる農地など) を表す地理情報データが必要であれば、その必要に合ったジオコーダーをプラグインすることもできます。

プラグイン・ジオコーダーを使用するには、まずその登録を済ませる必要があります。ユーザーおよびベンダーは、 db2gse.gse_register_gc ストアード・プロシージャーを使って登録を行えます。コントロール・センターから登録することはできません。 db2gse.gse_register_gc についての詳細は、96ページの『db2gse.gse_register_gc』を参照してください。 DB2 地理情報エクステンダーのストアード・プロシージャーの使用についての詳細は、 71ページの『第9章 ストアード・プロシージャー』を参照してください。

ジオコーダーは、以下の 2 つのモードで運用できます。

- *バッチ・モード* では、1 回の操作で、地理情報列用の既存ソース・データすべてを地理情報データに変換し、そのデータを地理情報列に入れる作業が行われます。この操作は、「ジオコーダーの実行 (Run Geocoder)」ウィンドウから開始できます。あるいは、db2gse.gse_run_gc ストアード・プロシージャーを呼び出すためにプログラムをコーディングして、アプリケーション・プログラムで開始することもできます。
- *増分モード* の場合、ジオコーダーは、表内でデータが挿入または更新されるときにそのデータを変換し、結果として得られる地理情報値を列に入れてその列を最新に保つようにします。それは、「地理情報レイヤーの作成 (Create Spatial Layer)」ウィンドウから要求できる挿入および更新トリガーを使って活動化できます。あるいは、db2gse.gse_enable_autogc ストアード・プロシージャーを呼び出すプログラムをコーディングして、アプリケーション・プログラムで要求することもできます。

  増分ジオコーディングは*自動ジオコーディング* とも呼ばれます。

ジオコーダーを使用するときには、以下の要素を考慮することができます。

1. コントロール・センターを使用するときは、「ジオコーダーの実行 (Run Geocoder)」ウィンドウを使用する前に、「地理情報レイヤーの作成 (Create Spatial Layers)」ウィンドウを使用するのが一般的です。つまり、バッチ・ジオコーディングを開始する前に、 DB2 地理情報エクステンダーから増分ジオコーディング用のトリガーを設定できます。したがって、増分ジオコーディングは、バッチ・ジオコーディングの前に実行することができます。すべてのソース・データをバッチ・モードで処理する場合、ジオコーダーは、増分モードで操作したのと同じデータをジオコーディングします。このような冗長性があっても、複製は行われません (地理情報データが 2 回生成されると、 2 番目のデータ生成が最初のものを上書きします)。ただし、パフ

ォーマンスは低下します。パフォーマンスの低下を回避する 1 つの方法として、バッチ・ジオコーディングが行われるまでトリガーの設定を据え置くことができます。

2. バッチ・モードでジオコーディングできるときにトリガーが配置されている場合は、バッチ・ジオコーディングが完了するまでトリガーを非活動化しておくことをお勧めします。そのようなトリガーは、「ジオコーダーの実行 (Run Geocoder)」ウィンドウから非活動化するか、db2gse.gse_disable_autogc ストアード・プロシージャーを呼び出すプログラムをコーディングすることにより、アプリケーション・プログラムから非活動化することができます。「ジオコーダーの実行 (Run Geocoder)」ウィンドウを使用する場合、DB2 地理情報エクステンダーは、ジオコーディングが完了した時点でそれらのトリガーを再び活動化します。db2gse.gse_disable_autogc ストアード・プロシージャーを使用する場合は、db2gse.gse_enable_autogc ストアード・プロシージャーを呼び出してトリガーを再び活動化できます。
3. 索引を持つ地理情報列にデータを取り込むためにジオコーダーをバッチ・モードで実行したい場合は、最初に索引を使用不可にするか除去してください。一方、ジオコーダーの実行中に索引を操作できる状態が続くと、パフォーマンスは極端に低下します。コントロール・センターを使用する場合、索引は、「ジオコーダーの実行 (Run Geocoder)」ウィンドウから使用不可にできます。 DB2 地理情報エクステンダーは、ジオコーディングが完了した時点で索引を自動的に再び使用可能にします。アプリケーション・プログラムを使用する場合、索引は、 SQL DROP ステートメントを使って除去できます。その場合は必ず、索引のパラメーターをメモに取っておいてください。そうすれば、バッチ・ジオコーディングの完了後に索引を再作成することができます。
4. ジオコーダーはソース・データのレコードを読み取るとき、そのレコードを参照データ内の対応レコードと一致させようとします。その一致は、ジオコーダーがレコードを処理できるように、一定の度合い (*精度* という) まで正確でなければなりません。たとえば、85 という精度は、ソース・レコードを処理するために、ソース・レコードと参照データ内の対応レコードの一致精度が、少なくとも 85 パーセントでなければならないことを意味します。

   精度の度合いは、ユーザーが指定します。精度は調整しなければならない場合もあるので注意してください。たとえば、精度が 100 であると仮定します。参照データよりも最新の住所を含むソース・レコードが多数ある場合、それらのレコードが参照データと 100 パーセントの精度で一致することは考えられません。結果として、ジオコーダーはそれらのレコードを拒否しま

す。ジオコーダーが不十分なデータや、かなり不正確なデータを生成した場合、そのような問題は一般に、精度を変更してジオコーダーを再実行すれば解決できる可能性があります。

ジオコーダーをバッチ・モードで実行する

ここでは、コントロール・センターからジオコーダーをバッチ・モードで実行する手順について概説します。その概説に続いて、各ステップを完了する方法を詳しく説明します。

ジオコーダーをバッチ・モードで実行するために必要な許可について調べるには、104ページの『許可』を参照してください。

ジオコーダーをバッチ・モードで実行するためのステップの概説:

1. 「ジオコーダーの実行 (Run Geocoder)」ウィンドウをオープンする。
2. 使用したいジオコーダーを指定する。
3. ジオコーダーのパフォーマンスを低下させる恐れがあるオブジェクトを使用不可にする。
4. DB2 がコミットを発行する前にジオコーディングするレコード数を指定する。
5. ジオコーダーの操作方法を指定する。
6. DB2 地理情報エクステンダーにジオコーダーの実行を命令する。

ジオコーダーをバッチ・モードで実行するための詳細なステップ:

1. 「ジオコーダーの実行 (Run Geocoder)」ウィンドウをオープンする。
   a. 「コントロール・センター (Control Center)」ウィンドウからオブジェクト・ツリーを拡張して、地理情報に対応したデータベース内の**「表 (Tables)」**フォルダーを表示する。
   b. **「表 (Tables)」**フォルダーをクリックする。ウィンドウの右側にある内容ペインに表が表示されます。
   c. 内容の中から任意の表を右マウス・ボタン・クリックし、ポップアップ・メニュー内の**「地理情報レイヤー (Spatial layers)」**をクリックする。「地理情報レイヤー (Spatial Layers)」ウィンドウがオープンします。
   d. 「地理情報レイヤー (Spatial Layers)」ウィンドウから、次のことを行ってください。
      1) データを入れる列に定義されているレイヤーを選択する。

2) **「ジオコーダーの実行 (Run Geocoder)」**押しボタンをクリックする。「ジオコーダーの実行 (Run Geocoder)」ウィンドウがオープンします。

2. デフォルト・ジオコーダーを使用したい場合は、**「名前 (Name)」**ボックス (このデフォルト名が表示されている) をそのままにしておく。それ以外の場合は、ボックスから任意のジオコーダーを選択します。
3. 以下の場合は、ジオコーダーのパフォーマンスを低下させる恐れがあるオブジェクトを使用不可にする。
   - データを入れる列に索引がある場合は、**「ジオコーディング・プロセス中は地理情報索引を一時的に使用不可にする (Temporarily disable spatial indexes during geocoding process)」**チェック・ボックスを選択します。
   - この列に増分ジオコーディングを活動化するトリガーが設定されている場合は、**「ジオコーディング・プロセス中は地理情報トリガーを一時的に使用不可にする (Temporarily disable spatial triggers during geocoding process)」**チェック・ボックスを選択します。

     索引およびトリガーは、「ジオコーダーの実行 (Run Geocoder)」ウィンドウの**「OK」**をクリックすると、自動的に再使用可能になります。
4. **「コミット効力範囲 (Commit scope)」**スピン・ボタンを使って、 DB2 がコミットを発行する前にジオコーディングするレコード数を指定する。たとえば、DB2 から、ジオコーディングした 100 レコードを一度にコミットするには、 100 という数値を指定します。

   ヒント: すべてのレコードが処理された後だけに DB2 からコミットを発行したい場合は、 0 を指定します。
5. **「ジオコーダー・パラメーター (Geocoder parameters)」**グループ・ボックスのフィールドを使って、ジオコーダーの操作方法を以下のように指定する。
   - **「精度レベル (Precision level)」**スピン・ボタンは、ソース・レコードと参照データの対応レコード間の精度をパーセントで指定するのに使用します。精度についての詳細は、43ページの『ジオコーディングについて』を参照してください。
   - ベンダーが提供するジオコーダーを使用し、その場合にサポートされる特性を使用したい場合は、**「特性 (Properties)」**ボックスを使ってそれらの特性を設定します。
   - 選択した表のサブセット行だけをジオコーディングしたい場合は、**「WHERE 文節 (WHERE clause)」**ボックスを使って、希望する行の基

準を指定する SELECT WHERE 文節をコーディングします。この文節は、表内の任意の列を参照できます。

基準だけを入力します。キーワード WHERE は省略します。たとえば、表に STATE 列があり、この列で値 MA を含む行だけをジオコーディングしたい場合は、次のように入力します。

```
STATE='MA'
```

6. 「**OK**」をクリックしてジオコーダーを実行する。

データのインポートとエクスポート

ここでは、データのインポートおよびエクスポートのプロセスを示し、コントロール・センターを使って以下のことを行う方法を説明します。

- データ交換ファイルから新規または既存の表にデータをインポートする
- データ交換ファイルから既存の表にデータをインポートする
- 表からデータ交換ファイルにデータをエクスポートする

インポートおよびエクスポートについて

以下に、地理情報データをインポートおよびエクスポートする理由を挙げます。また、エクスポート・ソースとインポート・ターゲットのインターフェースとなるデータ交換ファイルについても説明します。

DB2 地理情報エクステンダーを使えば、データ交換ファイルとの間で、地理情報データをインポートおよびエクスポートすることができます。以下のシナリオを考慮してください。

- GIS の中に、自分のオフィス、顧客、その他の業務関連事項を表す地理情報データが含まれている場合。このデータに、会社を取り巻く環境 (都市、道路、利益に資する場所など) を表す地理情報データを補足します。必要なデータは、地図のベンダーから入手できます。 DB2 地理情報エクステンダーを使えば、ベンダーが提供するデータ交換ファイルからデータをインポートできます。
- 地理情報データを Oracle システムから DB2 地理情報エクステンダー GIS に移行したい場合。作業は、Oracle ユーティリティーを使って、データをデータ交換ファイルにロードして行います。次に、DB2 地理情報エクステンダーを使って、このファイルから、地理情報操作を使用可能にしたデータベースにデータをインポートします。
- GIS ブラウザーを使って、地理情報を顧客にビジュアルに表示したい場合。ブラウザーには、作業元となるファイルだけが必要です。データベースに接続する必要はありません。 DB2 地理情報エクステンダーを使えば、データ

をデータ交換ファイルにエクスポートしてから、ブラウザー・ユーティリティーを使ってデータをブラウザーにロードすることができます。

コントロール・センターは DB2 地理情報エクステンダーで 2 種類のデータ交換ファイル、すなわち、形状ファイルと ESRI_SDE 転送ファイルをサポートします。形状ファイルは多くの場合、ファイル・システムで生成されるデータをインポートするためと、ファイル・システムにロードされるファイルにデータをエクスポートするために使用します。 ESRI_SDE 転送ファイルは多くの場合、 ESRI データベースで生成されるデータをインポートするために使用します。

データを新規または既存の表にインポートする

ここでは、形状ファイルまたは ESRI_SDE 転送ファイルからデータを新規または既存の表にインポートするためのステップを概説します。その概説に続いて、各ステップを完了する方法を詳しく説明します。

形状データのインポートに必要な許可を調べるには、 94ページの『許可』を参照してください。 ESRI_SDE データのインポートに必要な許可を調べるには、92ページの『許可』を参照してください。

新規または既存の表にデータをインポートするステップの概説:

1. 「地理情報データのインポート (Import Spatial Data)」ウィンドウをオープンする。
2. インポートされるデータが入ったファイルのパス、名前、および形式を指定する。
3. 各コミットの前にインポートするレコード数を指定する。
4. 作成される表に地理情報データをインポートしたい場合は、その表の名前、およびデータを入れる列の名前を指定する。地理情報データを既存の表にインポートする場合は、データを入れる列を指定する。
5. データに関連付けられる地理情報参照システムを指定する。
6. インポートに失敗したレコードを収集するファイルを指定する。
7. DB2 地理情報エクステンダーにデータのインポートを命令し、このウィンドウから表を定義した場合は、その表を作成し、データをレイヤーとして想定する列を登録する命令も与える。

データを新規または既存の表にインポートするステップの詳細:

1. 「地理情報データのインポート (Import Spatial Data)」ウィンドウをオープンする。

a. 「コントロール・センター (Control Center)」ウィンドウからオブジェクト・ツリーを展開して、 DB2 地理情報エクステンダーを実行するサーバーの下で**「データベース (Databases)」**フォルダーを見つける。

b. **「データベース (Databases)」**フォルダーをクリックする。ウィンドウの右側の内容ペインに、データベースが表示されます。

c. データをインポートするデータベースを右マウス・ボタン・クリックし、ポップアップ・メニューから**「地理情報エクステンダー (Spatial Extender)」** —> **「地理情報データのインポート (Import Spatial Data)」**をクリックする。「地理情報データのインポート (Import Spatial Data)」ウィンドウがオープンします。

2. インポートされるデータが入ったファイルのパス、名前、および形式を指定する。

a. **「ファイル名 (File name)」**フィールドを使って、パスと名前を指定する。

b. **「ファイル形式 (File format)」**ボックスを使って形式を指定する。指定できる形式は、次のとおりです。

Shape デフォルトはこの形式です。

ESRI_SDE

この形式を指定すると、**「地理情報参照名 (Spatial reference name)」**フィールドが、この形式に関連した地理情報参照システムの名前にデフォルト設定されます。

3. **「コミット効力範囲 (Commit scope)」**フィールドを使って、各コミットの前にインポートするレコード数を指定する。たとえば、DB2 から 1 回に 100 レコードをコミットするには、100 という数値を指定します。

ヒント: すべてのレコードが処理された後だけに DB2 からコミットを発行したい場合は、 0 を指定します。

4. データを入れる表と列を指定する。

a. **「レイヤー・スキーマ (Layer schema)」**ボックスを使って、データのインポート先となる表のスキーマを指定する。

b. 表および列を以下のように指定する。

- 表がまだ存在していない場合:

1) **「レイヤー表 (Layer table)」**フィールドに、表の名前を入力する。

2) **「レイヤー列 (Layer column)」**フィールドに、インポートされたデータが入る列の名前を入力する。 DB2 地理情報エクステンダーはこの列をレイヤーとして自動的に登録します。

- 表がすでに存在する場合:

1) **「レイヤー表 (Layer table)」**フィールドで表を指定する。この表には、インポートされたデータを入れる列がなければなりません。また、この列はレイヤーとして登録しておく必要もあります。

2) **「レイヤー列 (Layer column)」**フィールドで、インポートされたデータの対象となる列の名前を指定する。

5. **「地理情報参照名 (Spatial reference name)」**フィールドで、このデータに関連付けられる地理情報参照システムを入力または選択する。 (データの形式が ESRI_SDE 転送ファイルの場合、関連する地理情報参照システムはフィールドに自動的に表示されます。)
6. **「例外ファイル (Exception file)」**フィールドで、インポートに失敗したレコードを収集できる新規ファイルのパスと名前を指定する。これらのレコードは、後ほど修正し、このファイルからインポートできます。

   DB2 地理情報エクステンダーはこのファイルを作成します。既存のファイルは指定しないでください。
7. **「OK」**をクリックしてデータをインポートする。また、まだ存在していない表の名前を指定すると、この表が作成され、データの対象となる列がレイヤーとして登録されます。さらに、指定した例外ファイルも作成されます。

データを既存の表にインポートする

ここでは、形状ファイルまたは ESRI_SDE 転送ファイルからデータを既存の表にインポートするためのステップを概説します。その概説に続いて、各ステップを完了する方法を詳しく説明します。

形状データのインポートに必要な許可を調べるには、 94ページの『許可』を参照してください。 ESRI_SDE データのインポートに必要な許可を調べるには、92ページの『許可』を参照してください。

既存の表にデータをインポートするステップの概説:

1. 「地理情報データのインポート (Import Spatial Data)」ウィンドウをオープンする。
2. インポートされるデータが入ったファイルのパスと名前を指定する。
3. 各コミットの前にインポートするレコード数を指定する。
4. インポートする地理情報データが含められる列を指定する。

5. このデータに関連付けられる地理情報参照システムを指定する。
6. インポートに失敗したレコードを収集するファイルを指定する。
7. DB2 地理情報エクステンダーにデータのインポートを命令し、まだ作成されていない列を指定した場合は、その列を作成してレイヤーとして登録するよう命令を与える。

既存の表にデータをインポートするステップの詳細:

1. 「地理情報データのインポート (Import Spatial Data)」ウィンドウをオープンする。
   a. 「コントロール・センター (Control Center)」ウィンドウからオブジェクト・ツリーを展開して、データをインポートするデータベースの**「表 (Tables)」**フォルダーを見つける。
   b. **「表 (Tables)」**フォルダーをクリックする。ウィンドウの右側にある内容ペインに表が表示されます。
   c. データをインポートする表を右マウス・ボタン・クリックし、ポップアップ・メニューから**「地理情報エクステンダー (Spatial Extender)」** —> **「地理情報データのインポート (Import Spatial Data)」**をクリックする。「地理情報データのインポート (Import Spatial Data)」ウィンドウがオープンします。
2. **「ファイル名 (File name)」**ボックスで、インポートされるデータが入ったファイルのパスと名前を指定する。
3. **「コミット効力範囲 (Commit scope)」**ボックスを使って、各コミットの前にインポートするレコード数を指定する。たとえば、DB2 から 1 回に 100 レコードをコミットするには、100 という数値を指定します。

   ヒント: すべてのレコードが処理された後だけに DB2 からコミットを発行したい場合は、 0 を指定します。
4. インポートする地理情報データが含められる列を指定する。
   - 表に列がまだ存在していなければ、**「レイヤー列 (Layer column)」**ボックスを使って列の名前を入力する。
   - 列がすでに存在していれば、**「レイヤー列 (Layer column)」**ボックスを使って列の名前を選択または入力する。
5. **「地理情報参照名 (Spatial reference name)」**ボックスを使って、インポートされたデータに関連付けられる地理情報参照システムを指定する。
   - 表に列を追加する場合、地理情報参照システムの名前を入力または選択する。

- インポートしたデータが既存の列を対象にしている場合、**「地理情報参照名 (Spatial reference name)」**ボックスは現状のままにしておく。デフォルト地理情報参照システムの名前が表示されます。

6. **「例外ファイル (Exception file)」**フィールドで、インポートに失敗したレコードを収集できる新規ファイルのパスと名前を指定する。これらのレコードは、後ほど修正し、このファイルからインポートできます。

   DB2 地理情報エクステンダーはこのファイルを作成します。既存のファイルは指定しないでください。

7. **「OK」**をクリックしてデータをインポートする。また、まだ存在していない列を指定した場合、その列が作成され、レイヤーとして登録されます。さらに、指定した例外ファイルも作成されます。

データを形状ファイルにエクスポートする

ここでは、データを形状ファイルにエクスポートするためのステップを概説します。その概説に続いて、各ステップを完了する方法を詳しく説明します。

これらのステップを実行するのに必要な許可を調べるには、 90ページの『許可』を参照してください。

形状ファイルからデータをエクスポートするステップの概説:

1. 「地理情報データのエクスポート (Export Spatial Data)」ウィンドウをオープンする。
2. エクスポートされる地理情報データが入っている列を指定する。
3. データのサブセット行をエクスポートしたい場合は、そのサブセットを DB2 地理情報エクステンダーに識別させる。
4. データのエクスポート先となるファイルのパスと名前を指定する。
5. DB2 地理情報エクステンダーにデータのエクスポートを命令する。

形状ファイルにデータをエクスポートするステップの詳細:

1. 「地理情報データのエクスポート (Export Spatial Data)」ウィンドウをオープンする。
   a. 「コントロール・センター (Control Center)」ウィンドウからオブジェクト・ツリーを展開して、地理情報データを含むデータベースの**「表 (Tables)」**または**「視点 (Views)」**フォルダーを表示する。
   b. **「表 (Tables)」**または**「視点 (Views)」**フォルダーをクリックする。ウィンドウの右側の内容ペインに、表または視点が表示されます。
   c. エクスポートされるデータを含む表または視点を右マウス・ボタン・クリックし、ポップアップ・メニューの**「地理情報エクステンダー**

(Spatial Extender)」 —> **「地理情報データのエクスポート (Export Spatial Data)」**をクリックする。「地理情報データのエクスポート (Export Spatial Data)」ウィンドウがオープンします。

2. **「レイヤー列 (Layer column)」**フィールドで、エクスポートされる地理情報データが入っている列の名前を指定する。
3. 表のサブセット行をエクスポートしたい場合は、**「WHERE 文節 (WHERE clause)」**ボックスを使って、希望する行の基準を指定する WHERE 文節を入力する。この文節では、データのエクスポート元となる表または視点の列だけを参照できます。

   基準だけを入力します。キーワード WHERE は省略します。たとえば、表または視点に STATE 列があり、この列で値 MA を含む行だけをジオコーディングしたい場合は、次のように入力します。

   ```
   STATE='MA'
   ```

4. **「ファイル名 (File name)」**フィールドで、データのエクスポート先となるファイルのパスと名前を指定する。
5. **「OK」**をクリックしてデータをエクスポートする。

第6章 地理情報索引の作成

この章では、コントロール・センターを使って地理情報データの索引を作成する方法について説明します。

地理情報列にデータを入れたら、地理情報索引を作成することができます。 B ツリーなどの一般的な索引付け構造は、表データに対して線形の 1 次ソートを実行します。地理情報操作のために使用可能にされた表データは単一項目ではなく、 2 次元項目として格納されます。たとえば、ポリゴンなどの地理情報図形は、 1 つの地理情報列またはレイヤーにある、いくつかの座標値で構成されます。 B ツリー索引は地理情報データ・タイプを扱えないので、DB2 地理情報エクステンダーでは、*格子索引* という独自の索引付けテクノロジーを作成しました。格子索引は B ツリー索引に基づいていますが、2 次元データを処理し、地理情報列の索引付けを実行する機能が強化されています。格子索引は 3 つのレイヤーをサポートし、広範囲にわたるオブジェクト、サイズ、およびデータ分散において良好なパフォーマンスを実現できるようになっています。地理情報索引についての詳細は、123ページの『第12章 地理情報索引』を参照してください。

地理情報索引の作成に必要な許可を調べるには、84ページの『許可』を参照してください。

コントロール・センターを使って地理情報索引を作成する

コントロール・センターを使って地理情報索引を作成するには、次のようにします。

1. オブジェクト・ツリーで、**「表 (Tables)」**フォルダーを選択する。内容ペインに既存の表がすべて表示されます。
2. 内容ペインから、索引を作成する表を右マウス・ボタン・クリックし、ポップアップ・メニューの**「地理情報エクステンダー (Spatial Extender)」**—> **「地理情報索引 (Spatial Indexes)」**をクリックする。「地理情報索引 (Spatial Indexes)」ウィンドウがオープンします。
3. 「地理情報索引 (Spatial Indexes)」ウィンドウから**「作成 (Create)」**をクリックする。「地理情報索引の作成 (Create Spatial Index)」ウィンドウがオープンします。
4. **「名前 (Name)」**フィールドに、作成したい新規地理情報索引の名前を入力する。

注: スキーマを指定する必要はありません。 DB2 地理情報エクステンダーはスキーマを自動的に追加し、完全修飾名を作成します。

5. **「レイヤー列 (Layer column)」**フィールドで、索引作成の対象となるレイヤーを選択する。

   レイヤーとは、DB2 地理情報エクステンダーに対して定義または登録された地理情報列です。

6. **「格子サイズ (Grid size)」**フィールドに、各フィールドに割り当てたい格子サイズ・フィールド値を入力する。

   格子レベルである**「最上 (Finest)」**、**「中間 (Middle)」**、および**「粗大 (Coarsest)」**は、セル・サイズを増やすことによって入力します。したがって、2 番目のレベルのセル・サイズは最初のものよりも大きく、 3 番目のレベルのセル・サイズは 2 番目のものよりも大きくなります。

格子セル・サイズの判別

正確な格子サイズの判別は、試行錯誤を繰り返して行います。格子サイズは、索引作成の対象となるオブジェクトのおよそのサイズに合わせて設定することをお勧めします。格子サイズの設定は小さ過ぎても大き過ぎても、パフォーマンスの低下を招く恐れがあります。サイズが小さ過ぎると、索引検索時のキー / オブジェクト比率に影響します。そうなると、作成されるキーが多過ぎて、大量の候補が戻されます。格子サイズの設定が大き過ぎると、最初の索引検索で戻される候補は少数ですが、最後の表走査でパフォーマンスが低下する可能性があります。

格子セル・サイズの選択および格子レベルの数についての詳細は、 130ページの『格子セル・サイズの選択』を参照してください。

第7章 地理情報の検索と分析

地理情報索引を構成したら、地理情報表を使用する準備が整います。この章では、地理情報データの検索および分析に関連した論題を取り上げます。たとえば、さまざまな検索方法を概説し、地理情報関数を使用する表照会の例を示します。

地理情報分析の実行方法

地理情報分析を実行するには、以下のいずれかのプログラミング環境で SQL および地理情報関数を使用します。

- 地理情報ブラウザー (たとえば、ESRI の ArcExplorer)。

  ArcExplorer の使用方法については、 ESRI Web サイト http://www.esri.com から入手できる *Using ArcExplorer* を参照してください。
- 対話式 SQL ステートメント。
- ユーザー開発アプリケーション (たとえば、ODBC、JDBC、および組み込み SQL)。

アプリケーションは、DB2 コマンド・センター、DB2 コマンド・ウィンドウ、またはコマンド行プロセッサーから立ち上げることができます。

地理情報照会の作成

ここでは、地理情報の関数と述部を使用する地理情報照会の作成について説明します。

地理情報の関数と SQL

DB2 地理情報エクステンダーには、地理情報データに対してさまざまな操作を実行する関数が組み込まれています。ここに挙げる例では、地理情報関数を使って独自の地理情報照会を作成する方法を示します。

表3 は、地理情報の関数と、それらの関数が実行できる操作のタイプを示したリストです。

表 3. 地理情報の関数と操作

関数のタイプ	操作例
計算	2 地点間の距離を計算する

表 3. 地理情報の関数と操作 (続き)

関数のタイプ	操作例
比較	洪水地帯に位置する全顧客を検出する
データ交換	サポートされる形式にデータを変換する
変形	特定の地点から半径 5 マイルの範囲を追加する

地理情報関数についての詳細は、 133ページの『第13章 図形および関連する地理情報関数』および 169ページの『第14章 SQL 照会のための地理情報関数』を参照してください。

例 1: 比較

以下の照会は、各デパートからの平均的な顧客距離を検出します。この例で使用する地理情報関数は ST_Distance および ST_Within です。

```
SELECT s.id, AVG(db2gse.ST_Distance(c.location,s.location))
FROM customers c, stores s
WHERE db2gse.ST_Within(c.location,s.zone)=1
GROUP BY s.id
```

例 2: データ交換

以下の照会は、サンフランシスコ湾岸域に住む顧客の位置を検出します。この例で使用する地理情報関数は ST_AsText (データ交換) および ST_Within です。 ST_AsText は、c.location 列の地理情報データを OGC TEXT 形式に変換します。

```
SELECT db2gse.ST_AsText(c.location,cordref(1))
FROM customers c
WHERE db2gse.ST_Within(c.location,:BayArea)=1
```

例 3: 計算

以下の照会は、10.5 マイル以上の長さの道路すべてを検出します。この例で使用する地理情報関数は ST_Length です。

```
SELECT s.name,s.id
FROM street s
WHERE db2gse.ST_Length(s.path) > 10.5
```

例 4: 変形

この照会は、洪水地帯、または洪水地帯の境界から 2 マイル以内に住む顧客を検出します。この例で使用する地理情報関数は ST_Buffer (変形) および ST_Within です。

```
SELECT c.name,c.phoneNo,c.address
FROM customers c
WHERE db2gse.ST_Within(c.location,ST_Buffer(:floodzone,2))=1
```

地理情報の述部と SQL

地理情報述部と呼ばれる特殊な地理情報関数のグループを使うと、照会のパフォーマンスを向上させることができます。 2 つのポリゴンを比較して重なり具合いを調べる地理情報述部 (ST_Overlaps など) は、時間と記憶域の両要件から見て、実行の費用が高価になる恐れがあります。したがって、実行費用を最小化するための最適化技法が重要になります。 DB2 照会最適化プログラムは、地理情報索引を使って、この節で後述する規則に従って地理情報述部を使用するときの照会パフォーマンスを向上させます。地理情報述部についての詳細は、147ページの『述部関数』を参照してください。地理情報索引を活用するために使用する地理情報述部は、次のとおりです。

- ST_Contains
- ST_Crosses
- ST_Disjoint
- ST_Distance
- ST_Envelope
- ST_Equals
- ST_Intersects
- ST_Overlaps
- ST_Touches
- ST_Within

地理情報の関数と述部をすべて示した詳細なリストについては、 169ページの『第14章 SQL 照会のための地理情報関数』を参照してください。

索引活用の規則

以下の規則は、地理情報述部を使用して地理情報照会を最適化する場合に適用されます。

- 述部は WHERE 文節で使用しなければならない。
- 述部は比較の左辺に置かなければならない。たとえば次のようにします。

  ```
  WHERE db2gse.ST_Within(c.location,:BayArea)=1
  ```

- 等価性の比較では、整数定数 1 を使用しなければならない。

  ```
  WHERE db2gse.ST_Within(c.location,:BayArea)=1
  ```

- 検索ターゲットとして述部に使用する地理情報列がなければならず、その列に対する地理情報索引が作成されていなければならない。

索引活用の例

表4 は、地理情報索引を活用するために地理情報照会を作成する場合の正しい方法と間違った方法を示しています。

表 4. 索引活用の規則

地理情報照会	違反している規則

`SELECT *`
`FROM customers c`
`WHERE db2gse.ST_Within(c.location,:BayArea)=1`
	この例で違反している条件はありません。

`SELECT *`
`FROM customers c`
`WHERE db2gse.ST_Distance(c.location,:SanJose)<10`
	この例で違反している条件はありません。

`SELECT *`
`FROM customers c`
`WHERE db2gse.ST_Length(c.location)>10`
	述部は WHERE 文節で使用しなければならない。 (ST_Length は地理情報関数だが、述部ではない。)

`SELECT *`
`FROM customers c`
`WHERE 1=db2gse.ST_Within(c.location,:BayArea)`
	述部は比較の左辺に置かなければならない。

`SELECT *`
`FROM customers c`
`WHERE db2gse.ST_Within(c.location,:BayArea)=2`
	等価性の比較では、整数定数 1 を使用しなければならない。

`SELECT *`
`FROM customers c`
`WHERE db2gse.ST_Within(:SanJose,:BayArea)=1`
	検索ターゲットとして述部に使用する地理情報列がなければならず、その列に対する地理情報索引が作成されていなければならない。 (SanJose と BayArea は地理情報列ではないので、地理情報索引が関連付けられない。)

第8章 DB2 地理情報エクステンダー用のアプリケーションの作成

この章では、DB2 地理情報エクステンダーのサンプル・プログラムを使って、地理情報を処理およびカスタマイズするためのアプリケーションを作成する方法について説明します。含まれるトピックは、次のとおりです。

- サンプル・プログラムの使用
- サンプル・プログラムのステップ

サンプル・プログラムの使用

DB2 地理情報エクステンダーのサンプル・プログラムを使うと、アプリケーション・プログラミングが容易になります。サンプル・プログラムを使って、次のことを行えます。

- ルーチン地理情報プロシージャーを自動化する。
- サンプル・コードを独自アプリケーションへカット・アンド・ペーストする。
- 地理情報対応データベースを作成し保守するために一般に必要とされるステップを理解する。

サンプル・プログラムは、DB2 地理情報エクステンダー用の複雑なタスクのコーディング、たとえば、 DB2 地理情報エクステンダー・ストアード・プロシージャーを呼び出すためのデータベース・インターフェースを使用するアプリケーションの作成などのために使用します。サンプル・プログラムから、アプリケーションをコピーしてカスタマイズすることができます。 DB2 地理情報エクステンダーのプログラミング・ステップに精通していない場合、サンプル・プログラムを実行すれば、各ステップを詳細に見ることができます。しかし、まず最初にサンプル・データベースを作成する必要があります。これは、サンプルの makefile で行うことができます。サンプル・プログラムの作成および実行の方法については、 21ページの『インストールの検査』を参照してください。

サンプル・プログラムのステップ

表5 は、サンプル・プログラムのステップ、関連するストアード・プロシージャー、および各ステップの説明を示しています。ストアード・プロシージャーを呼び出すための C 関数は、 表5 のアクションの列に括弧付きで表示してあります。ストアード・プロシージャーについての詳細は、71ページの『第9章 ストアード・プロシージャー』を参照してください。サンプル・プログラムは、13ページの『シナリオ: 保険会社が自社の GIS を更新する場合』で紹介されているシナリオに基づいています。

表 5. DB2 地理情報エクステンダーのサンプル・プログラム

サンプル・プログラムのステップ	アクション	説明
地理情報データベースを使用可 / 使用不可にする	1. 地理情報データベースを使用可能にする (gseEnableDB) 2. 地理情報データベースを使用不可にする (gseDisableDB) 3. 地理情報データベースを使用可能にする (gseEnableDB)	1. これは、DB2 地理情報エクステンダーを使用するために必要な最初のステップです。地理情報操作用として使用可能になったデータベースには、地理情報タイプのセット、地理情報関数のセット、地理情報述部のセット、新規索引タイプ、管理表および管理視点のセットが備わっています。 2. このステップは通常、間違ったデータベースに対して地理情報機能を使用可能にした場合に実行します。地理情報データベースを使用不可にする場合、地理情報タイプのセット、地理情報関数のセット、地理情報述部のセット、新規索引タイプ、管理表および管理視点のセットを除去してください。 **注:** データベースの使用可能プロシージャーで作成されたオブジェクトに依存する作成済みオブジェクトがある場合、データベースを使用不可にすることはできません。たとえば、タイプ ST_Point の地理情報列を持つ表を作成すると、データベースは使用不可になりません。その表は、データベースの使用不可プロシージャーによって除去されるタイプ ST_Point に依存しているからです。 3. 1 と同様。

表 5. DB2 地理情報エクステンダーのサンプル・プログラム (続き)

サンプル・プログラムのステップ	アクション	説明
地理情報参照システムを登録する	1. CUSTOMERS 表の LOCATION 列について地理情報参照システムを登録する (gseEnableSref) 2. OFFICES 表の LOCATION 列について地理情報参照システムを登録する (gseEnableSref) 3. OFFICES 表の LOCATION 列について地理情報参照システムを抹消する (gseDisableSref) 4. OFFICES 表の ZONE 列について地理情報参照システムを再登録する (gseEnableSref)	1. このステップは、CUSTOMERS 表の地理情報データを解釈するために使用される新規地理情報参照システム (SRS) を定義します。地理情報参照システムには、地理情報対応データベースの列に格納できる形式で図形データが組み込まれます。 SRS が特定のレイヤーに登録された後、そのレイヤーに適用可能な座標は、関連する CUSTOMERS 表列に格納できます。 2. このステップは、OFFICES レイヤーの地理情報データを解釈するために使用される新規地理情報参照システム (SRS) を定義します。 各表レイヤーには、SRS を定義しておく必要があります。 OFFICES 表レイヤーは、CUSTOMERS 表レイヤーとは異なる関連 SRS を必要とする場合があります。 3. このステップは、レイヤーまたは地理情報列に間違った SRS パラメーターを指定した場合に実行します。 OFFICES 表レイヤーの SRS を抹消するときは、その定義ならびに関連パラメーターを除去してください。 4. このステップは、OFFICES レイヤーの地理情報データを解釈するために使用される新規地理情報参照システム (SRS) を定義します。

表 5. DB2 地理情報エクステンダーのサンプル・プログラム (続き)

サンプル・プログラムのステップ	アクション	説明
地理情報表を作成する	1. LOCATION 列を追加して CUSTOMERS 表を更新する (gseSetupTables) 2. OFFICES 表を作成する (gseSetupTables)	1. CUSTOMERS 表は、何年かの間にデータベースに格納された業務データを表します。ALTER TABLE ステートメントはタイプ ST_Point の新規列 (LOCATION) を追加します。この列には、後続ステップで住所列をジオコーディングすることによってデータが入れられます。 2. OFFICES 表は、データの中でも特に、保険会社の各支所の販売地域を表します。表全体には、後続ステップで DB2 以外のデータベースから属性データが入れられます。このステップには、SHAPE ファイルから OFFICES 表に属性データをインポートすることが関係しています。
地理情報レイヤーを登録する	1. CUSTOMERS 表の LOCATION 列をレイヤーとして登録する (gseRegisterLayer) 2. OFFICES 表の ZONE 列をレイヤーとして登録する (gseRegisterLayer)	これらのステップは、LOCATION および ZONE 列をレイヤーとして DB2 地理情報エクステンダーに登録します。 DB2 地理情報エクステンダー・ユーティリティー (たとえば、ジオコーダー) で地理情報列にデータを取り込んだりアクセスしたりするには、該当する列をレイヤーとして登録する必要があります。

表 5. DB2 地理情報エクステンダーのサンプル・プログラム (続き)

サンプル・プログラムのステップ	アクション	説明
地理情報レイヤーにデータを取り込む	1. CUSTOMERS 表の LOCATION 列の住所データをジオコーディングする (gseRunGC) 2. 追加モードを使って OFFICES 表をロードする (gseImportShape) 3. 作成モードを使って HAZARD_ZONE 表をロードする (gseImportShape)	1. このステップは、ジオコーダー・ユーティリティーを呼び出して、バッチ・ジオコーディングを実行します。バッチ・ジオコーディングは通常、表のかなりの部分をジオコーディングまたは再びジオコーディングする必要がある場合に実行します。 2. このステップは、OFFICES 表に、 SHAPE ファイルの形式で存在する地理情報データをロードします。 OFFICES 表が存在し、OFFICES/ZONE レイヤーが登録されているので、ロード・ユーティリティーは既存表に新規レコードを追加します。 3. このステップは、HAZARD_ZONE レイヤーに、 SHAPE ファイルの形式で存在する地理情報データをロードします。表およびレイヤーが存在しないので、ロード・ユーティリティーは表を作成し、データがロードされる前にレイヤーを登録します。
地理情報索引を使用可能にする	1. CUSTOMERS 表の LOCATION 列の地理情報索引を使用可能にする (gseEnableIdx) 2. OFFICES 表の ZONE 列の地理情報索引を使用可能にする (gseEnableIdx) 3. OFFICES 表の LOCATION 列の地理情報索引を使用可能にする (gseEnableIdx) 4. HAZARD_ZONE 表の BOUNDRY 列の地理情報索引を使用可能にする (gseEnableIdx)	これらのステップは、CUSTOMERS、OFFICES、および HAZARD_ZONE 表の地理情報索引を使用可能にします。
自動ジオコーディングを使用可能にする	1. CUSTOMERS 表の LOCATION および ADDRESS 列の自動ジオコーディングを使用可能にする (gseEnableAutoGC)	このステップは、ジオコーダーの自動呼び出しをオンにします。自動ジオコーディングを使用すると、 CUSTOMERS 表の LOCATION および ADDRESS 列を互いに同期させて、後続の挿入および更新操作を行うことができます。

表 5. DB2 地理情報エクステンダーのサンプル・プログラム (続き)

サンプル・プログラムのステップ	アクション	説明
CUSTOMERS 表に対して挿入 / 更新する	1. 異なる町名を持つ特定のレコードを挿入する (gseInsDelUpd) 2. 新しい住所を持つ特定のレコードを更新する (gseInsDelUpd)	これらのステップは、CUSTOMERS 表の LOCATION 列に対する挿入および更新を示します。自動ジオコーディングが使用可能になると、ADDRESS 列からの情報は、 LOCATION 列の中で挿入または更新されるときに自動的にジオコーディングされます。このプロセスは、前のステップで使用可能にされています。
自動ジオコーディングを使用不可にする	1. CUSTOMERS レイヤーの自動ジオコーディングを使用不可にする (gseDisableAutoGC) 2. CUSTOMERS レイヤーの地理情報索引を使用不可にする (gseDisableIdxCustomersLayer)	これらのステップはジオコーダーおよび地理情報索引の自動呼び出しを使用不可にし、次のステップの準備を行います (次のステップには、CUSTOMERS 表全体の再ジオコーディングが関係しています)。大量の地理データをロードする場合は、データをロードする前に地理情報索引を使用不可にした後、ロードが完了してから地理情報索引を使用可能にすることをお勧めします。
CUSTOMERS 表を再びジオコーディングする	1. CUSTOMERS レイヤーを低い精度レベル (100% ではなく 90%) で再びジオコーディングする (gseRunGC) 2. CUSTOMERS レイヤーの地理情報索引を再び使用可能にする (gseEnableIdx) 3. 自動ジオコーディングを低い精度レベル (100% ではなく 90%) で再び使用可能にする (gseEnableAutoGC)	これらのステップは、ジオコーダーをバッチ・モードで再び実行し、新しい精度レベルで自動ジオコーディングを再び使用可能にし、地理情報索引および自動ジオコーディングを再び使用可能にします。この処置は、地理情報管理者がジオコーディング・プロセスの失敗率が高いことに気付いた場合にお勧めします。精度レベルを 100% に設定すると、参照データ内に一致する住所が見つからないために、住所のジオコーディングが失敗する恐れがあります。精度レベルを下げることにより、ジオコーダーが一致するデータを見つける確率は高くなります。表を再びバッチ・モードでジオコーディングしたら、自動ジオコーディングと地理情報索引の両方を再び使用可能にして、後続の挿入および更新を行うために地理情報索引と地理情報列の増分保守を行えるようにします。

表 5. DB2 地理情報エクステンダーのサンプル・プログラム (続き)

サンプル・プログラムのステップ	アクション	説明
視点を作成し、その地理情報列を視点レイヤーとして登録する	1. CUSTOMERS 表と HAZARD_ZONE 表の結合に基づいて視点 HIGHRISK_CUSTOMERS を作成する (gseCreateView) 2. 視点の地理情報列を視点レイヤーとして登録する (gseRegisterLayer)	これらのステップは、視点を作成し、その地理情報列を視点レイヤーとして登録します。
地理情報分析を実行する	1. 各オフィスからの平均顧客距離を検出する (ST_Within、ST_Distance) 2. オフィスごとに顧客の平均的な所得と保険料を検出する (ST_Within) 3. 既存オフィスの管轄下にいない顧客を検出する (ST_Within) 4. 各オフィスの管轄範囲が重なり合う危険地帯の数を検出する (ST_Overlaps) 5. 特定顧客の位置から最も近いオフィスを検出する (そのオフィスはオフィス地域の中心部に位置すると仮定する) (ST_Distance、ST_Centroid) 6. 特定の危険地帯の境界に近い位置の顧客を検出する (ST_Buffer、ST_Overlaps) 7. 特定のオフィスの管轄下にある、リスクの高い顧客を検出する (すべてのステップは gseRunSpatialQueries を使用する)	これらのステップは、DB2 SQL 言語で地理情報の述部と関数を使って地理情報分析を実行します。 DB2 照会最適化プログラムは、地理情報列の地理情報索引を活用して、可能な限り、照会パフォーマンスを向上させます。

表 5. DB2 地理情報エクステンダーのサンプル・プログラム (続き)

サンプル・プログラムのステップ	アクション	説明
地理情報レイヤーをファイルにエクスポートする	highRiskCustomers レイヤーをエクスポートする (gseExportShape)	このステップでは、SHAPE ファイルに対する照会結果をエクスポートする例を示します。照会結果を別のファイル形式にエクスポートすると、その情報を第三者のツール (たとえば、ESRI ArcInfo) で使用できるようになります。

第2部 参照資料

第9章 ストアード・プロシージャー

この章では、DB2 地理情報エクステンダーで地形情報を構築するためのストアード・プロシージャーについて解説します。コントロール・センターからDB2 地理情報エクステンダーを使用可能にして使用する場合、これらのストアード・プロシージャーは暗黙的に呼び出します。たとえば、DB2 地理情報エクステンダーのウィンドウから**「OK」**をクリックすると、 DB2 はそのウィンドウに関連付けられたストアード・プロシージャー (複数の場合もある) を自動的に呼び出します。あるいは、ストアード・プロシージャーをアプリケーション・プログラムで明示的に呼び出すこともできます。そのようなプログラムにはヘッダー・ファイル db2gse.h を組み込むことをお勧めします。このファイルには、ストアード・プロシージャーのパラメーターに割り当てる定数のマクロ定義が入っています。 AIX の場合、この定義は $DB2INSTANCE/sqllib/include/ ディレクトリーに格納されています。 Windows NT では、%DB2PATH%¥include¥ ディレクトリーに格納されています。

重要:

ストアード・プロシージャーの入力パラメーターのための文字ストリングの定数は、すべて大文字です。これらの定数を必要とするパラメーターについては、この章の中の表に示されています。

ストアード・プロシージャーを呼び出す前に、 DB2 地理情報エクステンダーがインストールされているデータベースに暗黙的または明示的に接続しておく必要があります。使用する最初のストアード・プロシージャーは db2gse.gse_enable_db です。これにより、地理情報操作用にデータベースが使用可能になります。その他のストアード・プロシージャーは、データベースを使用可能にした後でのみ使用できます。

ストアード・プロシージャーのインプリメンテーションは、 DB2 地理情報エクステンダー・サーバー上の db2gse ライブラリーにアーカイブされます。

以下のリストを使って、名前別または実行されるタスク別にストアード・プロシージャーを検索できます。最初に示すのは、名前のリストです。

- 78ページの『db2gse.gse_disable_sref』
- 79ページの『db2gse.gse_enable_autogc』
- 83ページの『db2gse.gse_enable_db』
- 84ページの『db2gse.gse_enable_idx』
- 87ページの『db2gse.gse_enable_sref』
- 90ページの『db2gse.gse_export_shape』
- 92ページの『db2gse.gse_import_sde』
- 94ページの『db2gse.gse_import_shape』
- 96ページの『db2gse.gse_register_gc』
- 98ページの『db2gse.gse_register_layer』
- 104ページの『db2gse.gse_run_gc』
- 106ページの『db2gse.gse_unregist_gc』
- 107ページの『db2gse.gse_unregist_layer』

次のリストは、ストアード・プロシージャーが実行するタスクを示しています。

- 地理情報列の索引を作成する (84ページの『db2gse.gse_enable_idx』を参照)。
- 地理情報参照システムを作成する (87ページの『db2gse.gse_enable_sref』を参照)。
- 地理情報列と対応する属性列の同期化を自動的に保てないように、ジオコーダーを使用不可にする (74ページの『db2gse.gse_disable_autogc』を参照)。
- データベース内の地理情報操作のサポートを使用不可にする (77ページの『db2gse.gse_disable_db』を参照)。
- 地理情報参照システムを除去する (78ページの『db2gse.gse_disable_sref』を参照)。
- 地理情報操作をサポートできるようにデータベースを使用可能にする (83ページの『db2gse.gse_enable_db』を参照)。
- 地理情報列と対応する属性列の同期化を自動的に保つためにジオコーダーを使用可能にする (79ページの『db2gse.gse_enable_autogc』を参照)。
- レイヤーと関連表を形状ファイルにエクスポートする (90ページの『db2gse.gse_export_shape』を参照)。
- ESRI_SDE 転送ファイルからレイヤーと関連表をインポートする (92ページの『db2gse.gse_import_sde』を参照)。
- 形状ファイルからレイヤーと関連表をインポートする (94ページの『db2gse.gse_import_shape』を参照)。

- デフォルトのジオコーダー以外のジオコーダーを登録する (96ページの『db2gse.gse_register_gc』を参照)。
- 地理情報列をレイヤーとして登録する (98ページの『db2gse.gse_register_layer』を参照)。
- ジオコーダーをバッチ・モードで実行する (106ページの『db2gse.gse_unregist_gc』を参照)。
- デフォルトのジオコーダー以外のジオコーダーを抹消する (107ページの『db2gse.gse_unregist_layer』を参照)。
- レイヤーを抹消する (107ページの『db2gse.gse_unregist_layer』を参照)。

これらのタスクを実行できる順序についての詳細は、 3ページの『第1章 DB2 地理情報エクステンダーについて』および 61ページの『第8章 DB2 地理情報エクステンダー用のアプリケーションの作成』を参照してください。

db2gse.gse_disable_autogc

このストアード・プロシージャーは、地理情報列と関連属性列 (複数可) の同期を保つトリガーを除去したり、一時的に使用不可にする場合に使用します。たとえば、属性列 (複数可) の値をバッチ・モードでジオコーディングするときにはトリガーを使用不可にすることをお勧めします。この点についての詳細は、43ページの『ジオコーディングについて』を参照してください。

このストアード・プロシージャーを呼び出す場合のコードの例については、サンプル・プログラムの C 関数 gseDisableAutoGc を参照してください。このプログラムについての詳細は、61ページの『第8章 DB2 地理情報エクステンダー用のアプリケーションの作成』を参照してください。

許可

このストアード・プロシージャーが呼び出されるユーザー ID には、権限、特権、または特権のセットの形で権限が与えられていなければなりません。具体的には以下の権限が必要です。

- 除去または一時的に使用不可にされるトリガーが定義された表を含むデータベースに対する SYSADM または DBADM 権限。
- この表に対する CONTROL 特権。
- この表に対する ALTER、SELECT、および UPDATE 特権。

入力パラメーター

表 6. db2gse.gse_disable_autogc ストアード・プロシージャーの入力パラメーター。

名前	データ・タイプ	説明
operMode	SMALLINT	トリガーを除去するか、または一時的に使用不可にするかを指定する。 このパラメーターはヌルにはできません。 **注釈:** トリガーを除去するには、GSE_AUTOGC_DROP マクロを使用します。 トリガーを一時的に使用不可にするには、GSE_AUTOGC_INVALIDATE マクロを使用します。これらのマクロに関連した値を調べるには、db2gse.h ファイルを参照してください。 AIX の場合、このファイルは $DB2INSTANCE/sqllib/include/ ディレクトリーに格納されています。 Windows NT では、%DB2PATH%¥include¥ ディレクトリーに格納されています。
layerSchema	VARCHAR(30)	layerTable パラメーターで指定された表または視点が属するスキーマの名前。 このパラメーターはヌル可能です。 **注釈:** layerSchema パラメーターの値を指定しない場合、 db2gse.gse_disable_autogc ストアード・プロシージャーを呼び出すために使用するユーザー ID がデフォルト設定されます。
layerTable	VARCHAR(128)	除去または一時的に使用不可にするトリガーが定義されている表の名前。 このパラメーターはヌルにはできません。
layerColumn	VARCHAR(128)	除去または一時的に使用不可にしたいトリガーによって保守される地理情報使用可能列の名前。 このパラメーターはヌルにはできません。

出力パラメーター

表 7. db2gse.gse_disable_autogc ストアード・プロシージャーの出力パラメーター。

名前	データ・タイプ	説明
msgCode	INTEGER	このストアード・プロシージャーの呼び出し元が戻せるメッセージに関連したコード。
予約済み	VARCHAR(1024)	完全なエラー・メッセージ (DB2 地理情報エクステンダー・サーバーで構成される)。

db2gse.gse_disable_db

このストアード・プロシージャーは、DB2 地理情報エクステンダーが地理情報データを格納し、そのデータに対する操作をサポートするためのリソースを除去する場合に使用します。

このストアード・プロシージャーの目的は、地理情報操作用のデータベースを使用可能にした後に、しかもそのデータベースに地理情報の表列またはデータを追加する*前に* 起こる種々の問題の解決を図ることにあります。たとえば、データベースを地理情報操作用に使用可能にした後に、 DB2 地理情報エクステンダーを別のデータベースに代用する場合があるかもしれません。地理情報列またはインポートした地理情報データを何も定義していない限り、このストアード・プロシージャーを呼び出して最初のデータベースからすべての地理情報リソースを除去することができます。

このストアード・プロシージャーを呼び出す場合のコードの例については、サンプル・プログラムの C 関数 gseDisableDB を参照してください。このプログラムについての詳細は、61ページの『第8章 DB2 地理情報エクステンダー用のアプリケーションの作成』を参照してください。

許可

このストアード・プロシージャーを呼び出すために使用するユーザー ID は、DB2 地理情報エクステンダー・リソースが除去されるデータベースに対する SYSADM または DBADM 権限を持っている必要があります。

出力パラメーター

表 8. db2gse.gse_disable_db ストアード・プロシージャーの出力パラメーター。

名前	データ・タイプ	説明
msgCode	INTEGER	このストアード・プロシージャーの呼び出し元が戻せるメッセージに関連したコード。
予約済み	VARCHAR(1024)	完全なエラー・メッセージ (DB2 地理情報エクステンダー・サーバーで構成される)。

db2gse.gse_disable_sref

このストアード・プロシージャーは、地理情報参照システムを除去するために使用します。このストアード・プロシージャーが処理されるとき、地理情報参照システムについての情報は DB2GSE.SPATIAL_REF_SYS カタログ視点から除去されます。この視点についての詳細は、121ページの『DB2GSE.SPATIAL_REF_SYS』を参照してください。

このストアード・プロシージャーを呼び出す場合のコードの例については、サンプル・プログラムの C 関数 gseDisableSref を参照してください。このプログラムについての詳細は、61ページの『第8章 DB2 地理情報エクステンダー用のアプリケーションの作成』を参照してください。

許可

不要です。

入力パラメーター

表 9. db2gse.gse_disable_sref ストアード・プロシージャーの入力パラメーター。

名前	データ・タイプ	説明
srId	INTEGER	除去される地理情報参照システムの数値識別子。 このパラメーターはヌルにはできません。

出力パラメーター

表 10. db2gse.gse_disable_sref ストアード・プロシージャーの出力パラメーター。

名前	データ・タイプ	説明
msgCode	INTEGER	このストアード・プロシージャーの呼び出し元が戻せるメッセージに関連したコード。
予約済み	VARCHAR(1024)	完全なエラー・メッセージ (DB2 地理情報エクステンダー・サーバーで構成される)。

制約事項

地理情報参照システムを除去するには、そのシステムを使用しているレイヤーをまず抹消しなければなりません。そのようなレイヤーが未登録のままであると、地理情報参照システムを除去する要求は拒否されます。

db2gse.gse_enable_autogc

このストアード・プロシージャーは、以下のことを行うために使用します。

- 地理情報列と関連属性列 (複数可) の同期を保つトリガーを作成する。属性列で値が挿入または更新されるたびに、トリガーは登録されているジオコーダーを使って、挿入または更新された値をジオコーディングし、その結果として得られたデータを地理情報列に入れます。
- 一時的に使用不可にされた後にトリガーを再活動化する。
- 挿入または更新された値をジオコーディングするために使用する関数を設定する。

このストアード・プロシージャーを呼び出す場合のコードの例については、サンプル・プログラムの C 関数 gseEnableAutoGC を参照してください。このプログラムについての詳細は、61ページの『第8章 DB2 地理情報エクステンダー用のアプリケーションの作成』を参照してください。

許可

このストアード・プロシージャーが呼び出されるユーザー ID には、権限、特権、または特権のセットの形で権限が与えられていなければなりません。具体的には以下の権限が必要です。

- このストアード・プロシージャーで作成したトリガーが定義されている表を含むデータベースに対する SYSADM または DBADM 権限。
- この表に対する CONTROL 特権。
- この表に対する ALTER、SELECT、および UPDATE 特権。

入力パラメーター

表 11. db2gse.gse_enable_autogc ストアード・プロシージャーの入力パラメーター。

名前	データ・タイプ	説明
operMode	SMALLINT	ジオコーディングを開始するトリガーを初めて作成するのか、それとも一時的に使用不可であった状態から再活動化するのかを指定する値。 このパラメーターはヌルにはできません。 **注釈:** トリガーを作成するには、GSE_AUTOGC_CREATE マクロを使用してください。トリガーを再活動化するには、GSE_AUTOGC_RECREATE マクロを使用します。これらのマクロに関連した値を調べるには、db2gse.h ファイルを参照してください。 AIX の場合、このファイルは $DB2INSTANCE/sqllib/include/ ディレクトリーに格納されています。 Windows NT では、%DB2PATH%¥include¥ ディレクトリーに格納されています。
layerSchema	VARCHAR(30)	layerTable パラメーターで指定された表が属するスキーマの名前。 このパラメーターはヌル可能です。 **注釈:** layerSchema パラメーターの値を指定しない場合、 db2gse.gse_enable_autogc ストアード・プロシージャーを呼び出すために使用するユーザー IDがデフォルト設定されます。
layerTable	VARCHAR(128)	このストアード・プロシージャーによって作成または再活動化されるトリガーが操作する表の名前。 このパラメーターはヌルにはできません。
layerColumn	VARCHAR(128)	このストアード・プロシージャーが作成または再活動化するトリガーによって保守される地理情報列の名前。 このパラメーターはヌルにはできません。

表 11. db2gse.gse_enable_autogc ストアード・プロシージャーの入力パラメーター。 (続き)

名前	データ・タイプ	説明
gcId	INTEGER	このストアード・プロシージャーが作成または再活動化する挿入または更新トリガーによって呼び出されるジオコーダーの識別子。 このパラメーターは、operMode パラメーターが GSE_AUTOGC_CREATE に設定されている場合には、ヌルにはできません。 operMode が GSE_AUTOGC_RECREATE に設定されている場合はヌル可能です。
precisionLevel	INTEGER	ジオコーダーがソース・データを正常に処理するために、ソース・データを対応する参照データと一致させる度合い。 このパラメーターは、operMode パラメーターが GSE_AUTOGC_CREATE に設定されている場合には、ヌルにはできません。 operMode が GSE_AUTOGC_RECREATE に設定されている場合はヌル可能です。 **注釈:** 精度レベルは、1 ～ 100 パーセントの範囲内で指定できます。
vendorSpecific	VARCHAR(256)	ベンダーによって提供される技術情報。たとえば、ベンダーがパラメーターの設定に使用するファイルのパスと名前。 このパラメーターは、operMode パラメーターが GSE_AUTOGC_CREATE に設定されている場合には、ヌルにはできません。 operMode が GSE_AUTOGC_RECREATE に設定されている場合はヌル可能です。

出力パラメーター

表 12. db2gse.gse_enable_autogc ストアード・プロシージャーの出力パラメーター。

名前	データ・タイプ	説明
msgCode	INTEGER	このストアード・プロシージャーの呼び出し元が戻せるメッセージに関連したコード。
予約済み	VARCHAR(1024)	完全なエラー・メッセージ (DB2 地理情報エクステンダー・サーバーで構成される)。

制約事項

- layerColumn パラメーターは、表レイヤーとして登録されている列を参照しなければなりません。
- operMode パラメーターが GSE_AUTOGC_CREATE に設定されている場合、登録されたジオコーダーの識別子を gcId パラメーターに割り当てる必要があります。

db2gse.gse_enable_db

このストアード・プロシージャーは、地理情報データを格納し、操作をサポートするために必要なリソースを、データベースに提供するために使用します。これらのリソースには、地理情報データ・タイプ、地理情報索引タイプ、カタログ表とカタログ視点、提供される関数、その他のストアード・プロシージャーが含まれています。

このストアード・プロシージャーを呼び出す場合のコードの例については、サンプル・プログラムの C 関数 gseEnableDB を参照してください。このプログラムについての詳細は、61ページの『第8章 DB2 地理情報エクステンダー用のアプリケーションの作成』を参照してください。

許可

このストアード・プロシージャーを呼び出すために使用するユーザー ID は、使用可能にされるデータベースに対する SYSADM または DBADM 権限を持っている必要があります。

出力パラメーター

表 13. db2gse.gse_enable_db ストアード・プロシージャーの出力パラメーター。

名前	データ・タイプ	説明
msgCode	INTEGER	このストアード・プロシージャーの呼び出し元が戻せるメッセージに関連したコード。
予約済み	VARCHAR(1024)	完全なエラー・メッセージ (DB2 地理情報エクステンダー・サーバーで構成される)。

db2gse.gse_enable_idx

このストアード・プロシージャーは、地理情報列の索引を作成するために使用します。

このストアード・プロシージャーを呼び出す場合のコードの例については、サンプル・プログラムの C 関数 gseEnableIdx を参照してください。このプログラムについての詳細は、61ページの『第8章 DB2 地理情報エクステンダー用のアプリケーションの作成』を参照してください。

許可

このストアード・プロシージャーを呼び出す場合に使用するユーザー ID は、以下のいずれかの権限または特権を持っている必要があります。

- 使用可能にされた索引を使用する表を含んだデータベースに対する SYSADM または DBADM権限。
- この表に対する CONTROL または INDEX 特権。

入力パラメーター

表 14. db2gse.gse_enable_idx ストアード・プロシージャーの入力パラメーター。

名前	データ・タイプ	説明
layerSchema	VARCHAR(30)	layerTable パラメーターで指定された表が属するスキーマの名前。 このパラメーターはヌル可能です。 **注釈:** layerSchema パラメーターの値を指定しない場合、 db2gse.gse_enable_idx ストアード・プロシージャーを呼び出すために使用するユーザー ID がデフォルト設定されます。
layerTable	VARCHAR(128)	作成する索引が定義される表の名前。 このパラメーターはヌルにはできません。
layerColumn	VARCHAR(128)	作成する索引を活用して検索される使用可能な地理情報列の名前。 このパラメーターはヌルにはできません。

表 14. db2gse.gse_enable_idx ストアード・プロシージャーの入力パラメーター。 (続き)

名前	データ・タイプ	説明
indexName	VARCHAR(128)	作成される索引の名前。 このパラメーターはヌルにはできません。 **注釈:** スキーマ名は使用しないでください。 DB2 地理情報エクステンダーは、layerSchema パラメーターによって参照されるスキーマに索引を自動的に割り当てます。
gridSize1	DOUBLE	最適な索引格子の細かさを指定する数値。 このパラメーターはヌルにはできません。
gridSize2	DOUBLE	(1) この索引の 2 番目の格子を設定しないか、(2) 2 番目の格子をどれほどの細かさにするかを表す数値。 このパラメーターはヌル可能です。 **注釈:** 2 番目の格子を設定しない場合は、0 を指定してください。 2 番目の格子が必要であれば、gridSize1 によって表示された格子よりも小さい値を指定する必要があります。
gridSize3	DOUBLE	(1) この索引の 3 番目の格子を設定しないか、(2) 3 番目の格子をどれほどの細かさにするかを表す数値。 このパラメーターはヌル可能です。 **注釈:** 3 番目の格子を設定しない場合は、0 を指定してください。 3 番目の格子が必要であれば、gridSize2 によって表示された格子よりも小さい値を指定する必要があります。

出力パラメーター

表 15. db2gse.gse_enable_idx ストアード・プロシージャーの出力パラメーター。

名前	データ・タイプ	説明
msgCode	INTEGER	このストアード・プロシージャーの呼び出し元が戻せるメッセージに関連したコード。
予約済み	VARCHAR(1024)	完全なエラー・メッセージ (DB2 地理情報エクステンダー・サーバーで構成される)。

db2gse.gse_enable_sref

このストアード・プロシージャーは、特定の座標系の負の数および小数を正の整数に変換し、 DB2 地理情報エクステンダーで格納できるようにする方法を指定する場合に使用します。ここで指定した内容を集合的に*地理情報参照システム* といいます。このストアード・プロシージャーが処理されるとき、地理情報参照システムについての情報は DB2GSE.SPATIAL_REF_SYS カタログ視点に追加されます。この視点についての詳細は、121ページの『DB2GSE.SPATIAL_REF_SYS』を参照してください。

このストアード・プロシージャーを呼び出す場合のコードの例については、サンプル・プログラムの C 関数 gseEnableSref を参照してください。このプログラムについての詳細は、61ページの『第8章 DB2 地理情報エクステンダー用のアプリケーションの作成』を参照してください。

許可

不要です。

入力パラメーター

表 16. db2gse.gse_enable_sref ストアード・プロシージャーの入力パラメーター。

名前	データ・タイプ	説明
srId	INTEGER	地理情報参照システムの数値識別子。 このパラメーターはヌルにはできません。 **注釈:**この識別子は、地理情報対応データベース内で固有でなければなりません。
srName	VARCHAR(64)	地理情報参照システムの短い説明。 このパラメーターはヌルにはできません。 **注釈:**この記述は、地理情報対応データベース内で固有でなければなりません。
falsex	DOUBLE	負の X 座標値から減算された場合に、負の数以外の数値 (すなわち、正の数またはゼロ) として残す数値。 このパラメーターはヌルにはできません。

表 16. db2gse.gse_enable_sref ストアード・プロシージャーの入力パラメーター。 (続き)

名前	データ・タイプ	説明
falsey	DOUBLE	負の Y 座標値から減算された場合に、負の数以外の数値 (すなわち、正の数またはゼロ) として残す数値。 このパラメーターはヌルにはできません。
xyunits	DOUBLE	小数の X 座標または小数の Y 座標で乗算されたときに、 32 ビット・データ項目として格納できる整数を生み出す数値。 このパラメーターはヌルにはできません。
falsez	DOUBLE	負の Z 座標値から減算された場合に、負の数以外の数値 (すなわち、正の数またはゼロ) として残す数値。 このパラメーターはヌルにはできません。
zunits	DOUBLE	小数の Z 座標で乗算されたときに、32 ビット・データ項目として格納できる整数を生み出す数値。 このパラメーターはヌルにはできません。
falsem	DOUBLE	負の測定値から減算された場合に、負の数以外の数値 (すなわち、正の数またはゼロ) として残す数値。 このパラメーターはヌルにはできません。
munits	DOUBLE	小数の測定値で乗算されたときに、32 ビット・データ項目として格納できる整数を生み出す数値。 このパラメーターはヌルにはできません。
scId	INTEGER	地理情報参照システムの導出元となっている座標系の数値識別子。座標系の数値識別子について知りたい場合は、 DB2GSE.COORD_REF_SYS カタログ視点 119ページの『DB2GSE.COORD_REF_SYS』を参照してください。 このパラメーターはヌルにはできません。

出力パラメーター

表 17. db2gse.gse_enable_sref ストアード・プロシージャーの出力パラメーター。

名前	データ・タイプ	説明
msgCode	INTEGER	このストアード・プロシージャーの呼び出し元が戻せるメッセージに関連したコード。
予約済み	VARCHAR(1024)	完全なエラー・メッセージ (DB2 地理情報エクステンダー・サーバーで構成される)。

db2gse.gse_export_shape

このストアード・プロシージャーは、レイヤーとその関連表を形状ファイルにエクスポートしたり、新しい形状ファイルを作って、その新規ファイルにレイヤーと関連表をエクスポートしたりする場合に使用します。

このストアード・プロシージャーを呼び出す場合のコードの例については、サンプル・プログラムの C 関数 gseExportShape を参照してください。このプログラムについての詳細は、61ページの『第8章 DB2 地理情報エクステンダー用のアプリケーションの作成』を参照してください。

許可

このストアード・プロシージャーを呼び出すために使用するユーザー ID は、エクスポートされる表に対する SELECT 特権を持っている必要があります。

入力パラメーター

表 18. db2gse.gse_export_shape ストアード・プロシージャーの入力パラメーター。

名前	データ・タイプ	説明
layerSchema	VARCHAR(30)	layerTable パラメーターで指定された表が属するスキーマの名前。 このパラメーターはヌル可能です。 **注釈:** layerSchema パラメーターの値を指定しない場合、 db2gse.gse_export_shape ストアード・プロシージャーを呼び出すために使用するユーザー ID がデフォルト設定されます。
layerTable	VARCHAR(128)	エクスポートする表の名前。 このパラメーターはヌルにはできません。
layerColumn	VARCHAR(30)	エクスポートするレイヤーとして登録されている列の名前。 このパラメーターはヌルにはできません。
fileName	VARCHAR(128)	指定したレイヤーのエクスポート先となる形状ファイルの名前。 このパラメーターはヌルにはできません。

表 18. db2gse.gse_export_shape ストアード・プロシージャーの入力パラメーター。 (続き)

名前	データ・タイプ	説明
whereClause	VARCHAR(1024)	SQL WHERE 文節の本体。ここでは、ジオコーディングされるレコード・セットに対する制限を定義します。その文節は、エクスポートする表の任意の属性列を参照できます。 このパラメーターはヌル可能です。

出力パラメーター

表 19. db2gse.gse_export_shape ストアード・プロシージャーの出力パラメーター。

名前	データ・タイプ	説明
msgCode	INTEGER	このストアード・プロシージャーの呼び出し元が戻せるメッセージに関連したコード。
予約済み	VARCHAR(1024)	完全なエラー・メッセージ (DB2 地理情報エクステンダー・サーバーで構成される)。

制約事項

1 回にエクスポートできるレイヤーは 1 つだけです。

db2gse.gse_import_sde

このストアード・プロシージャーは、地理情報操作で使用可能になったデータベースに SDE 転送ファイルをインポートするために使用します。このストアード・プロシージャーは、次の 2 つの方法のいずれかで操作できます。

- SDE 転送ファイルが登録済みレイヤー列を持つ既存表のターゲットになっている場合、 DB2 地理情報エクステンダーはその表にファイルのデータをロードします。
- それ以外の場合、DB2 地理情報エクステンダーは地理情報列を持つ表を作成して、その列をレイヤーとして登録し、そのレイヤーと表内の他の列にファイルのデータをロードします。

SDE 転送ファイルで指定された地理情報参照システムは、 DB2 地理情報エクステンダーに登録された地理情報参照システムと比較されます。指定されたシステムが登録済みシステムと一致する場合、転送データ内の負の数および小数の値はロード時に、登録済みシステムで規定された方法で変更されます。指定されたシステムが登録済みシステムのいずれとも一致しない場合、 DB2 地理情報エクステンダーは新規地理情報参照システムを作成して変更を規定します。

許可

既存表にデータをインポートする場合、このストアード・プロシージャーを呼び出す場合に使用するユーザー ID は、以下のいずれかの権限または特権を持っている必要があります。

- データがインポートされる表を含むデータベースに対する SYSADM または DBADM 権限。
- この表に対する CONTROL 特権。

データをインポートしたい表を作成しなければならない場合、このストアード・プロシージャーを呼び出す場合に使用するユーザー ID は、以下のいずれかの権限または特権を持っている必要があります。

- 作成される表を含むデータベースに対する SYSADM または DBADM 権限。

入力パラメーター

表 20. db2gse.gse_import_sde ストアード・プロシージャーの入力パラメーター。

名前	データ・タイプ	説明
layerSchema	VARCHAR(30)	layerTable パラメーターで指定された表または視点が属するスキーマの名前。 このパラメーターはヌル可能です。 **注釈:** layerSchema パラメーターの値を指定しない場合、 db2gse.gse_import_sde ストアード・プロシージャーを呼び出すために使用するユーザー ID がデフォルト設定されます。
layerTable	VARCHAR(128)	SDE 転送データのロード先となる表の名前。 このパラメーターはヌルにはできません。
layerColumn	VARCHAR(30)	SDE 転送ファイルの地理情報データのロード先となるレイヤーとして登録されている列の名前。 このパラメーターはヌルにはできません。
fileName	VARCHAR(128)	インポートされる SDE 転送ファイルの名前。 このパラメーターはヌルにはできません。
commitScope	INTEGER	チェックポイント当たりのレコード数。 このパラメーターはヌル可能です。

出力パラメーター

表 21. db2gse.gse_import_sde ストアード・プロシージャーの出力パラメーター。

名前	データ・タイプ	説明
msgCode	INTEGER	このストアード・プロシージャーの呼び出し元が戻せるメッセージに関連したコード。
予約済み	VARCHAR(1024)	完全なエラー・メッセージ (DB2 地理情報エクステンダー・サーバーで構成される)。

db2gse.gse_import_shape

このストアード・プロシージャーは、地理情報操作で使用可能になったデータベースに形状ファイルをインポートするために使用します。このストアード・プロシージャーは、次の 2 つの方法のいずれかで操作できます。

- 形状ファイルが登録済みレイヤー列を持つ既存表のターゲットになっている場合、 DB2 地理情報エクステンダーはその表にファイルのデータをロードします。
- それ以外の場合、DB2 地理情報エクステンダーは地理情報列を持つ表を作成して、その列をレイヤーとして登録し、そのレイヤーと表内の他の列にファイルのデータをロードします。

このストアード・プロシージャーを呼び出す場合のコードの例については、サンプル・プログラムの C 関数 gseImportShape を参照してください。このプログラムについての詳細は、61ページの『第8章 DB2 地理情報エクステンダー用のアプリケーションの作成』を参照してください。

許可

このストアード・プロシージャーを呼び出す場合に使用するユーザー ID は、以下のいずれかの権限または特権を持っている必要があります。

- インポートした形状データがロードされる表を含んだデータベースに対する SYSADM または DBADM 権限。
- この表に対する CONTROL 特権。

入力パラメーター

表 22. db2gse.gse_import_shape ストアード・プロシージャーの入力パラメーター。

名前	データ・タイプ	説明
layerSchema	VARCHAR(30)	layerTable パラメーターで指定された表または視点が属するスキーマの名前。 このパラメーターはヌル可能です。 **注釈:** layerSchema パラメーターの値を指定しない場合、 db2gse.gse_import_shape ストアード・プロシージャーを呼び出すために使用するユーザー ID がデフォルト設定されます。
layerTable	VARCHAR(128)	インポートした形状ファイルのロード先となる表の名前。 このパラメーターはヌルにはできません。

表 22. db2gse.gse_import_shape ストアード・プロシージャーの入力パラメーター。 (続き)

名前	データ・タイプ	説明
layerColumn	VARCHAR(30)	形状データのロード先となるレイヤーとして登録されている列の名前。 このパラメーターはヌルにはできません。
fileName	VARCHAR(128)	インポートされる形状ファイルの名前。 このパラメーターはヌルにはできません。
exceptionFile	VARCHAR(128)	インポートできなかった形状が格納されるファイルのパスと名前。これは、db2gse.gse_import_shape ストアード・プロシージャーが実行されるときに作成される新規ファイルです。 このパラメーターはヌルにはできません。
srId	INTEGER	形状データのロード先となるレイヤーに使用する地理情報参照システムの識別子。 このパラメーターはヌル可能です。 **注釈:** この識別子を指定しない場合、内部変換は、形状ファイルの解像度として可能な最大解像度に設定されます。
commitScope	INTEGER	チェックポイント当たりのレコード数。 このパラメーターはヌル可能です。

出力パラメーター

表 23. db2gse.gse_import_shape ストアード・プロシージャーの出力パラメーター。

名前	データ・タイプ	説明
msgCode	INTEGER	このストアード・プロシージャーの呼び出し元が戻せるメッセージに関連したコード。
予約済み	VARCHAR(1024)	完全なエラー・メッセージ (DB2 地理情報エクステンダー・サーバーで構成される)。

db2gse.gse_register_gc

このストアード・プロシージャーは、デフォルト以外のジオコーダーを登録するために使用します。ジオコーダーがすでに登録されているかどうかを調べるには、 DB2GSE.SPATIAL_GEOCODER カタログ視点 (120ページの『DB2GSE.SPATIAL_GEOCODER』に記述) を参照してください。

許可

このストアード・プロシージャーを呼び出すために使用するユーザー ID は、このストアード・プロシージャーが登録するジオコーダーを含んだデータベースに対する SYSADM または DBADM 権限を持っている必要があります。

入力パラメーター

表 24. db2gse.gse_register_gc ストアード・プロシージャーの入力パラメーター。

名前	データ・タイプ	説明
gcId	INTEGER	登録するジオコーダーの数値識別子。 このパラメーターはヌルにはできません。 **注釈:** この識別子は、データベース内で固有でなければなりません。
gcName	VARCHAR(64)	登録するジオコーダーの短い説明。 このパラメーターはヌルにはできません。 **注釈:** この記述は、データベース内で固有の文字ストリングでなければなりません。
vendorName	VARCHAR(64)	登録するジオコーダーを提供したベンダーの名前。 このパラメーターはヌルにはできません。
primaryUDF	VARCHAR(256)	登録するジオコーダーの完全修飾名。 このパラメーターはヌルにはできません。
precisionLevel	INTEGER	ジオコーダーがソース・データを正常に処理するために、ソース・データを対応する参照データと一致させる度合い。 このパラメーターはヌルにはできません。 **注釈:** 精度レベルは、1 ～ 100 パーセントの範囲内で指定できます。

表 24. db2gse.gse_register_gc ストアード・プロシージャーの入力パラメーター。 (続き)

名前	データ・タイプ	説明
vendorSpecific	VARCHAR(256)	ベンダーによって提供される技術情報。たとえば、ベンダーがパラメーターの設定に使用するファイルのパスと名前。 このパラメーターはヌル可能です。
geoArea	VARCHAR(256)	ジオコーディングされる地理上の領域。 このパラメーターはヌル可能です。
説明	VARCHAR(256)	ベンダーによって提供された注釈。 このパラメーターはヌル可能です。

出力パラメーター

表 25. db2gse.gse_register_gc ストアード・プロシージャーの出力パラメーター。

名前	データ・タイプ	説明
msgCode	INTEGER	このストアード・プロシージャーの呼び出し元が戻せるメッセージに関連したコード。
予約済み	VARCHAR(1024)	完全なエラー・メッセージ (DB2 地理情報エクステンダー・サーバーで構成される)。

db2gse.gse_register_layer

このストアード・プロシージャーは、地理情報列をレイヤーとして登録するために使用します。このストアード・プロシージャーが処理されるとき、登録されるレイヤーについての情報は DB2GSE.GEOMETRY_COLUMNS カタログ視点に追加されます。この視点についての詳細は、120ページの『DB2GSE.GEOMETRY_COLUMNS』を参照してください。

このストアード・プロシージャーを呼び出す場合のコードの例については、サンプル・プログラムの C 関数 gseRegisterLayer を参照してください。このプログラムについての詳細は、61ページの『第8章 DB2 地理情報エクステンダー用のアプリケーションの作成』を参照してください。

許可

このストアード・プロシージャーを呼び出す場合に使用するユーザー ID は、以下のいずれかの権限または特権を持っている必要があります。

- 表レイヤーの場合は、次のとおりです。
  - このレイヤーが属する表を含んだデータベースに対する SYSADM または DBADM 権限。
  - この表に対する CONTROL または ALTER 特権。
- 視点レイヤーの場合は、次のとおりです。
  - (1) このレイヤーのためにジオコーディングする住所データを含み、 (2) ジオコーディングの結果として得られる地理情報データを含んだ基礎表 (複数可) に対する SELECT 特権。

入力パラメーター

表 26. db2gse.gse_register_layer ストアード・プロシージャーの入力パラメーター。

名前	データ・タイプ	説明
layerSchema	INTEGER(30)	layerTable パラメーターで指定された表または視点が属するスキーマの名前。 このパラメーターはヌル可能です。 **注釈:** layerSchema パラメーターの値を指定しない場合、 db2gse.gse_register_layer ストアード・プロシージャーを呼び出すために使用するユーザー ID がデフォルト設定されます。

表 26. db2gse.gse_register_layer ストアード・プロシージャーの入力パラメーター。 (続き)

名前	データ・タイプ	説明
layerTable	VARCHAR(128)	レイヤーとして登録される列を含んだ表または視点の名前。 このパラメーターはヌルにはできません。
layerColumn	VARCHAR(128)	レイヤーとして登録される列の名前。列が存在しない場合、DB2 地理情報エクステンダーはその列を作成します。 このパラメーターはヌルにはできません。
layerTypeName	VARCHAR(64)	レイヤーとして登録される列のデータ・タイプ。データ・タイプは大文字で指定してください。たとえば、次のようにします。 `ST_POINT` このパラメーターは、列がこのストアード・プロシージャーの処理時に作成される表列である場合には、ヌルにできません。そうでない場合、列が表または視点内の既存の列であれば、このパラメーターはヌル可能です。
srId	INTEGER	このレイヤーに使用する地理情報参照システムの識別子。 このパラメーターは表レイヤーにはヌルにはできません。視点レイヤーを登録するときは、DB2 地理情報エクステンダーはこのパラメーターを無視します。
geoSchema	VARCHAR(30)	視点列をレイヤーとして登録するときに適用されます。 geoSchema パラメーターは、列が属する視点の基礎となる表のスキーマです。 レイヤーとして視点列を登録する場合、このパラメーターはヌル可能です。表列を登録するときは、DB2 地理情報エクステンダーはこのパラメーターを無視します。 **注釈:** geoSchema パラメーターの値を指定しない場合、デフォルトとして layerSchema パラメーターの値が設定されます。

表 26. db2gse.gse_register_layer ストアード・プロシージャーの入力パラメーター。 (続き)

名前	データ・タイプ	説明
geoTable	VARCHAR(128)	視点列をレイヤーとして登録するときに適用されます。 geoTable パラメーターは、列が属する視点の基礎となる表の名前です。 レイヤーとして視点列を登録する場合、このパラメーターはヌルにできません。表列を登録するときは、DB2 地理情報エクステンダーはこのパラメーターを無視します。
geoColumn	VARCHAR(128)	視点列をレイヤーとして登録するときに適用されます。 geoColumn パラメーターは、この視点列の基礎となる表列の名前です。 レイヤーとして視点列を登録する場合、このパラメーターはヌルにできません。表列を登録するときは、DB2 地理情報エクステンダーはこのパラメーターを無視します。
nAttributes	SMALLINT	このレイヤーのためにジオコーディングされるソース・データを含んだ列の数。 レイヤーとして表列を登録する場合、このパラメーターはヌル可能です。レイヤーとして視点列を登録するときは、DB2 地理情報エクステンダーはこのパラメーターを無視します。
attr1Name	VARCHAR(128)	このレイヤーのためにジオコーディングされるソース・データを含んだ最初の列の名前。 レイヤーとして表列を登録する場合、このパラメーターはヌル可能です。レイヤーとして視点列を登録するときは、DB2 地理情報エクステンダーはこのパラメーターを無視します。 デフォルトのジオコーダーを使用する場合は、attr1Name 列に番地を保管する必要があります。

表 26. db2gse.gse_register_layer ストアード・プロシージャーの入力パラメーター。 (続き)

名前	データ・タイプ	説明
attr2Name	VARCHAR(128)	このレイヤーのためにジオコーディングされるソース・データを含んだ 2 番目の列の名前。 レイヤーとして表列を登録する場合、このパラメーターはヌル可能です。レイヤーとして視点列を登録するときは、DB2 地理情報エクステンダーはこのパラメーターを無視します。 デフォルトのジオコーダーを使用する場合は、attr2Name 列に都市名を保管する必要があります。
attr3Name	VARCHAR(128)	このレイヤーのためにジオコーディングされるソース・データを含んだ 3 番目の列の名前。 レイヤーとして表列を登録する場合、このパラメーターはヌル可能です。レイヤーとして視点列を登録するときは、DB2 地理情報エクステンダーはこのパラメーターを無視します。 デフォルトのジオコーダーを使用する場合は、attr3Name 列に州 (都道府県) の名前または省略形を保管する必要があります。
attr4Name	VARCHAR(128)	このレイヤーのためにジオコーディングされるソース・データを含んだ 4 番目の列の名前。 レイヤーとして表列を登録する場合、このパラメーターはヌル可能です。レイヤーとして視点列を登録するときは、DB2 地理情報エクステンダーはこのパラメーターを無視します。 デフォルトのジオコーダーを使用する場合は、attr4Name 列に郵便番号を保管する必要があります。

表 26. *db2gse.gse_register_layer* ストアード・プロシージャーの入力パラメーター。 *(続き)*

名前	データ・タイプ	説明
attr5Name	VARCHAR(128)	このレイヤーのためにジオコーディングされるソース・データを含んだ 5 番目の列の名前。 レイヤーとして表列を登録する場合、このパラメーターはヌル可能です。レイヤーとして視点列を登録するときは、DB2 地理情報エクステンダーはこのパラメーターを無視します。 デフォルトのジオコーダーは Attr5Name 列を無視します。
attr6Name	VARCHAR(128)	このレイヤーのためにジオコーディングされるソース・データを含んだ 6 番目の列の名前。 レイヤーとして表列を登録する場合、このパラメーターはヌル可能です。レイヤーとして視点列を登録するときは、DB2 地理情報エクステンダーはこのパラメーターを無視します。 デフォルトのジオコーダーは Attr6Name 列を無視します。
attr7Name	VARCHAR(128)	このレイヤーのためにジオコーディングされるソース・データを含んだ 7 番目の列の名前。 レイヤーとして表列を登録する場合、このパラメーターはヌル可能です。レイヤーとして視点列を登録するときは、DB2 地理情報エクステンダーはこのパラメーターを無視します。 デフォルトのジオコーダーは Attr7Name 列を無視します。
attr8Name	VARCHAR(128)	このレイヤーのためにジオコーディングされるソース・データを含んだ 8 番目の列の名前。 レイヤーとして表列を登録する場合、このパラメーターはヌル可能です。レイヤーとして視点列を登録するときは、DB2 地理情報エクステンダーはこのパラメーターを無視します。 デフォルトのジオコーダーは Attr8Name 列を無視します。

表 26. db2gse.gse_register_layer ストアード・プロシージャーの入力パラメーター。 (続き)

名前	データ・タイプ	説明
attr9Name	VARCHAR(128)	このレイヤーのためにジオコーディングされるソース・データを含んだ 9 番目の列の名前。 レイヤーとして表列を登録する場合、このパラメーターはヌル可能です。レイヤーとして視点列を登録するときは、DB2 地理情報エクステンダーはこのパラメーターを無視します。 デフォルトのジオコーダーは Attr9Name 列を無視します。
attr10Name	VARCHAR(128)	このレイヤーのためにジオコーディングされるソース・データを含んだ 10 番目の列の名前。 レイヤーとして表列を登録する場合、このパラメーターはヌル可能です。レイヤーとして視点列を登録するときは、DB2 地理情報エクステンダーはこのパラメーターを無視します。 デフォルトのジオコーダーは Attr10Name 列を無視します。

出力パラメーター

表 27. db2gse.gse_register_layer ストアード・プロシージャーの出力パラメーター。

名前	データ・タイプ	説明
msgCode	INTEGER	このストアード・プロシージャーの呼び出し元が戻せるメッセージに関連したコード。
予約済み	VARCHAR(1024)	完全なエラー・メッセージ (DB2 地理情報エクステンダー・サーバーで構成される)。

制約事項

- レイヤーとして視点列を登録する場合、その列は、すでにレイヤーとして登録されている表列に基づいていなければなりません。
- 登録するレイヤーのためにジオコーディングするデータを 11 個以上の属性列に入れることはできません。

db2gse.gse_run_gc

このストアード・プロシージャーは、ジオコーダーをバッチ・モードで実行するために使用します。このタスクについての詳細は、46ページの『ジオコーダーをバッチ・モードで実行する』を参照してください。

このストアード・プロシージャーを呼び出す場合のコードの例については、サンプル・プログラムの C 関数 gseRunGC を参照してください。このプログラムについての詳細は、61ページの『第8章 DB2 地理情報エクステンダー用のアプリケーションの作成』を参照してください。

許可

このストアード・プロシージャーを呼び出す場合に使用するユーザー ID は、以下のいずれかの権限または特権を持っている必要があります。

- 指定されたジオコーダーが操作される表の入ったデータベースに対する SYSADM または DBADM 権限。
- この表に対する CONTROL または UPDATE 特権。

入力パラメーター

表 28. db2gse.gse_run_gc ストアード・プロシージャーの入力パラメーター。

名前	データ・タイプ	説明
layerSchema	VARCHAR(30)	layerTable パラメーターで指定された表または視点が属するスキーマの名前。 このパラメーターはヌル可能です。 **注釈:** layerSchema パラメーターの値を指定しない場合、 db2gse.gse_run_gc を呼び出すために使用するユーザー ID がデフォルト設定されます。
layerTable	VARCHAR(128)	ジオコーディングされたデータの挿入先となる列を含んだ表の名前。 このパラメーターはヌルにはできません。
layerColumn	VARCHAR(128)	ジオコーディングされたデータの挿入先となる列の名前。 このパラメーターはヌルにはできません。

表 28. db2gse.gse_run_gc ストアード・プロシージャーの入力パラメーター。 (続き)

名前	データ・タイプ	説明
gcId	INTEGER	実行したいジオコーダーの識別子。 このパラメーターはヌル可能です。 登録されたジオコーダーの識別子を調べるには、DB2GSE.SPATIAL_GEOCODER カタログ視点を参照してください。
precisionLevel	INTEGER	ジオコーダーがソース・データを正常に処理するために、ソース・データを対応する参照データと一致させる度合い。 このパラメーターはヌル可能です。 **注釈:** 精度レベルは、1 ～ 100 パーセントの範囲内で指定できます。
vendorSpecific	VARCHAR(256)	ベンダーによって提供される技術情報。たとえば、ベンダーがパラメーターの設定に使用するファイルのパスと名前。 このパラメーターはヌル可能です。
whereClause	VARCHAR(256)	WHERE 文節の本体。ここでは、ジオコーディングされるレコード・セットに対する制限を定義します。その文節は、ジオコーダーが操作する表の任意の属性列を参照できます。 このパラメーターはヌル可能です。
commitScope	INTEGER	チェックポイント当たりのレコード数。 このパラメーターはヌル可能です。

出力パラメーター

表 29. db2gse.gse_run_gc ストアード・プロシージャーの出力パラメーター。

名前	データ・タイプ	説明
msgCode	INTEGER	このストアード・プロシージャーの呼び出し元が戻せるメッセージに関連したコード。
予約済み	VARCHAR(1024)	完全なエラー・メッセージ (DB2 地理情報エクステンダー・サーバーで構成される)。

db2gse.gse_unregist_gc

このストアード・プロシージャーは、デフォルト・ジオコーダー以外のジオコーダーを抹消するために使用します。

抹消したいジオコーダーについての情報を調べるには、DB2GSE.SPATIAL_GEOCODER カタログ視点を参照してください。 120ページの『DB2GSE.SPATIAL_GEOCODER』を参照してください。

許可

このストアード・プロシージャーを呼び出すために使用するユーザー ID は、抹消するジオコーダーを含んだデータベースに対する SYSADM または DBADM 権限を持っている必要があります。

入力パラメーター

表 30. db2gse.gse_unregist_gc ストアード・プロシージャーの入力パラメーター。

名前	データ・タイプ	説明
gcId	INTEGER	抹消するジオコーダーの識別子。 このパラメーターはヌルにはできません。

出力パラメーター

表 31. db2gse.gse_unregist_gc ストアード・プロシージャーの出力パラメーター。

名前	データ・タイプ	説明
msgCode	INTEGER	このストアード・プロシージャーの呼び出し元が戻せるメッセージに関連したコード。
予約済み	VARCHAR(1024)	完全なエラー・メッセージ (DB2 地理情報エクステンダー・サーバーで構成される)。

db2gse.gse_unregist_layer

このストアード・プロシージャーは、レイヤーを抹消するために使用します。ストアード・プロシージャーは、次のようにして抹消を実行します。

- DB2 地理情報エクステンダーのカタログ表からレイヤーの定義を除去する。
- レイヤーの地理情報データがレイヤーの地理情報参照システムの要件に適合するように、 DB2 地理情報エクステンダーがこのレイヤーの基礎表に課した検査制約を削除する。
- 住所データが追加、変更、または除去されるたびに地理情報列を更新するために使用するトリガーを除去する。

このストアード・プロシージャーが処理されるとき、レイヤーについての情報は DB2GSE.GEOMETRY_COLUMNS メタ視点から除去されます。この視点についての詳細は、120ページの『DB2GSE.GEOMETRY_COLUMNS』を参照してください。

許可

このストアード・プロシージャーを呼び出す場合に使用するユーザー ID は、以下のいずれかの権限または特権を持っている必要があります。

- 表レイヤーの場合は、次のとおりです。
  - このレイヤーの基礎表を含んだデータベースに対する SYSADM または DBADM 権限。
  - この表に対する CONTROL または ALTER 特権。
- 視点レイヤーの場合は、次のとおりです。
  - (1) このレイヤーのためにジオコーディングする住所データを含み、 (2) ジオコーディングの結果として得られる地理情報データを含んだ基礎表 (複数可) に対する SELECT 特権。

入力パラメーター

表 32. db2gse.gse_unregist_layer ストアード・プロシージャーの入力パラメーター。

名前	データ・タイプ	説明
layerSchema	VARCHAR(30)	layerTable パラメーターで指定された表が属するスキーマの名前。 このパラメーターはヌル可能です。 **注釈:** layerSchema パラメーターの値を指定しない場合、 db2gse.gse_unregister_layer ストアード・プロシージャーを呼び出すために使用するユーザー ID がデフォルト設定されます。
layerTable	VARCHAR(128)	layerColumn パラメーターに指定された列を含んだ表の名前。 このパラメーターはヌルにはできません。
layerColumn	VARCHAR(128)	抹消したいレイヤーとして定義されている地理情報列の名前。 このパラメーターはヌルにはできません。

出力パラメーター

表 33. db2gse.gse_unregist_layer ストアード・プロシージャーの出力パラメーター。

名前	データ・タイプ	説明
msgCode	INTEGER	このストアード・プロシージャーの呼び出し元が戻せるメッセージに関連したコード。
予約済み	VARCHAR(1024)	完全なエラー・メッセージ (DB2 地理情報エクステンダー・サーバーで構成される)。

制約事項

視点レイヤーとして定義されている視点列が、表レイヤーとして定義されている表列に基づいている場合、視点レイヤーを抹消するまでこの表レイヤーを抹消することはできません。

第10章 メッセージ

この章では、DB2 地理情報エクステンダーがユーザーに戻すメッセージについて解説します。各メッセージには、識別子があります。 E の文字で終わる識別子はエラー・メッセージ、W で終わる識別子は警告、 I で終わる識別子は一般的な情報を示しています。

DBA7200E More than 10 columns are selected as input to a geocoder

説明: 10 個までの列を選択してジオコーダーに入力できます。

ユーザーの応答: 「選択済みの列 (Selected columns)」ボックスにリストされる名前が 10 個以下になるまで、同ボックスから列名を「使用可能な列 (Available columns)」ボックスに移動してください。

DBA7201E The database is not enabled for Spatial Extender operation.

説明: 地理情報エクステンダーを使用する前に、地理情報エクステンダー用にデータベースを使用可能にする必要があります。

ユーザーの応答: データベースを右マウス・ボタン・クリックし、メニューから「地理情報エクステンダー (Spatial Extender)」 ->「使用可能にする (Enable)」を選択してください。

GSE0000I The operation is completed successfully.

GSE0001E Spatial Extender could not perform the requested operation (“<operation-name>”) under user ID “<user-id>”.

説明: 要求した操作のユーザー ID は、操作を実行する特権または権限を持っていません。

ユーザーの応答: 資料を参照して適切な許可を調べるか、地理情報エクステンダー管理者から適切な権限を入手してください。

GSE0002E “<value>” is not a valid value for the “<argument-name>” argument.

説明: 入力した値は間違っているか、つづりが誤っています。

ユーザーの応答: 資料を参照するか、地理情報エクステンダー管理者に連絡して、指定する必要のある値または値の範囲を調べてください。

GSE0003E Spatial Extender could not perform the requested operation because argument “<argument-name>” was not specified.

説明: この操作に必須の引き数を指定しませんでした。

ユーザーの応答: 引き数 “<argument-name>” に値を指定してから、操作を再要求してください。

GSE0004W The argument “<argument-name>” was not evaluated.

説明: 要求した操作は引き数 “<argument-name>” を使用しません。

ユーザーの応答: 不要です。

GSE0005E Spatial Extender could not process your request to create an object named "<object-name>".

説明: オブジェクト "<object-name>" がすでに存在するか、そのオブジェクトを作成するための適切な許可を持っていません。オブジェクトは、表、列、トリガー、索引、ファイル、または他の種類のオブジェクトのいずれかです。

ユーザーの応答: "<object-name>" が希望するオブジェクトであって存在する場合、何も行う必要はありません。そうでない場合は、名前を正確に指定し、オブジェクトを作成する適正な許可を持っていることを確かめてください。

GSE0006E Spatial Extender could not perform the requested operation on an enabled or registered object named "<object-name>".

説明: オブジェクト "<object-name>" はすでに使用可能か登録されている、またはすでに存在します。オブジェクトは、レイヤー、索引、地理情報参照システム、座標系、ジオコーダー、または他の種類のオブジェクトのいずれかです。

ユーザーの応答: オブジェクト "<object-name>" が存在することを確かめてから、要求を再実行依頼してください。

GSE0007E Spatial Extender could not perform the requested operation on "<object-name>", an object that has not yet been enabled or registered.

説明: オブジェクト "<object-name>" は使用可能になっていないか登録されていません。オブジェクトは、レイヤー、索引、地理情報参照システム、地理情報座標システム、ジオコーダー、または他の種類のオブジェクトのいずれかです。

ユーザーの応答: オブジェクト "<object-name>" を使用可能にするか登録してください。その後、要求を再実行依頼してください。

GSE0008E An unexpected SQL error ("<sql-error-message>") has occurred.

ユーザーの応答: SQL エラー・メッセージ "<sql-error-message>" の SQLCODE に関連付けられた詳細なメッセージを調べてください。必要に応じて、IBM 技術員に連絡してください。

GSE0009E The requested operation could not be performed on an object named "<object-name>" that already exists.

説明: "<object-name>" はすでにデータベースまたはオペレーティング・システムに存在しています。このオブジェクトは、ファイル、表、視点、列、索引、トリガー、または他の種類のオブジェクトのいずれかです。

ユーザーの応答: オブジェクトにアクセスしようとするときは、そのオブジェクトを正確に指定するようにしてください。オブジェクトは必要に応じて削除してください。

GSE0010E The requested operation could not be performed on an object named "<object-name>" that might not exist.

説明: "<object-name>" はデータベースまたはオペレーティング・システムに存在していません。このオブジェクトには、ファイル、表、視点、列、索引、トリガー、ファイル、または他の種類のオブジェクトのいずれかです。

ユーザーの応答: オブジェクトにアクセスするための適正な許可を持っていることを確かめてください。その許可を持っており、オブジェクトが存在しない場合は、そのオブジェクトを作成する必要があります。

GSE0011E Spatial Extender could not disable or unregister object “<object-name>”.

説明: “<object-name>” は別のオブジェクトに従属しています。 “<object-name>” は、地理情報参照システム、レイヤー、ジオコーダー、または他の種類のオブジェクトのいずれかです。

ユーザーの応答: 資料を参照して、“<object-name>” が従属する可能性のあるオブジェクトの種類を調べてください。その後、“<object-name>” が従属する特定のオブジェクトを除去してください。

GSE0012E Spatial Extender could not process your request because the fully qualified spatial column “<layer-schema.layer-name.layer-column>” is not registered as a table layer.

説明: 完全修飾された地理情報列 “<layer-schema.layer-name.layer-column>” を表レイヤーとして登録しなければ、関連する特定の操作 (たとえば、その索引を使用可能にすること、ジオコーダーを使用可能にしてバッチ・モードで取り込んだり、自動的に更新したりすること) を実行することはできません。

ユーザーの応答: 地理情報エクステンダー・カタログ内の DB2GSE.GEOMETRY_COLUMNS 視点を検査することにより、完全修飾された地理情報列 “<layer-schema.layer-name.layer-column>” が、レイヤーとして登録されていることを確認してください。また、この列が入っている表に、有効な対応する属性列が含まれていることも確かめてください。

GSE0013E The database is not enabled for spatial operations.

説明: データベースは地理情報操作用に使用可能になっていません。したがって、地理情報エクステンダー・カタログは存在しません。

ユーザーの応答: 地理情報操作用にデータベースを使用可能にしてください。

GSE0014E The database has already been enabled for spatial operations.

説明: データベースはすでに地理情報操作用に使用可能になっています。

ユーザーの応答: 期待通りにデータベースが使用可能になっていることを確かめてください。データベースは、必要に応じて使用不可にしてください。

GSE0498E The following error occurred: “<error-message>”.

GSE0499W Spatial Extender issued the following warning: “<warning-message>”.

GSE0500E The operation mode that you specified (“<operation-mode>”) is invalid.

説明: 指定したモードは、要求した操作ではサポートされていません。

ユーザーの応答: 資料を参照して、その操作でサポートされているモードを調べてください。

GSE1001E Spatial Extender was unable to register a view layer that is named “<schema-name.view-name.column-name>” and that is based on spatial column “<schema-name.table-name.column-name>”.

説明: 指定した地理情報列 (“<schema-name.table-name.column-name>”) は、表レイヤーとして登録されていません。

ユーザーの応答: 列 “<schema-name.table-name.column-name>” を表レイヤーとして登録してください。

GSE1002E Spatial Extender was unable to register a view layer that is named “<schema-name.view-name.column-name>” and that is based on table “<schema-name.table-name>”.

説明: 指定した表 (“<schema-name.table-name>”) は直接的にも間接的にも、視点 “<schema-name.view-name.column-name>” の基礎表ではありません。

ユーザーの応答: 視点 “<schema-name.view-name.column-name>” の基礎表を調べて、その表を指定してください。

GSE1003E Spatial Extender was unable to access a column named “<column-name>” in a table or view named “<schema-name.object-name>”.

説明: 表または視点 “<schema-name.object-name>” は、“<column-name>” という名前の列を持っていません。

ユーザーの応答: 表または視点 “<schema-name.object-name>” の定義を検査して、該当する列の正しい名前を調べてください。

GSE1004E Spatial Extender was unable to register the fully qualified spatial column “<schema-name.table-name.column-name>” as a table layer.

説明: 列 “<schema-name.table-name.column-name>” は、地理情報データ・タイプを持っていないか、基礎表と関連付けられていません。

ユーザーの応答: 列 “<schema-name.table-name.column-name>” に地理情報データ・タイプを定義するか、この列がローカル基礎表の一部であることを確かめてください。

GSE1005E The spatial reference system (“<view-layer-spatial-reference-id>”) that you specified for a view layer differs from the spatial reference system (“<table-layer-spatial-reference-id>”) that is used for this layer’s underlying table layer.

説明: 視点レイヤーの地理情報参照システムは、基礎表レイヤーの地理情報参照システムと同じでなければなりません。

ユーザーの応答: 視点レイヤー用に基礎表レイヤーの地理情報参照システムを指定してください。

GSE1006E Because “<spatial-reference-id>” is an invalid spatial reference system ID, Spatial Extender was unable to register the layer that you requested.

説明: 指定した地理情報参照システム (“<spatial-reference-id>”) は使用可能になっていないか登録されていません。

ユーザーの応答: 地理情報参照システムを使用可能にするか登録してください。その後、レイヤーを登録するための要求を再実行依頼してください。

GSE1007E An SQL error (SQLSTATE "<sqlstate>") might have occurred when Spatial Extender tried unsuccessfully to add a spatial column ("<column-name>") to table "<schema-name.table-name>".

ユーザーの応答: SQLSTATE "<sqlstate>" に関連付けられたメッセージを検索してください。

GSE1008E DB2 地理情報エクステンダー was unable to register a view layer"<layer-schema.layer-name.layer-column>" because the spatial data type "<layer-column-type>" of the view layer does not match the spatial data type "<geo-column-type>" of the underlying table layer "<geo-schema.geo-name.geo-column>".

説明: 視点レイヤー "<layer-schema.layer-name.layer-column>" の地理情報データ・タイプは、そのレイヤーの基礎表レイヤー "<geo-schema.geo-name.geo-column>" の地理情報データ・タイプと一致しなければなりません。これら 2 つのデータ・タイプの間に矛盾があると、地理情報データの処理時にあいまいさが生じます。

ユーザーの応答: 視点レイヤーの地理情報データ・タイプとその基礎表レイヤーが同じであることを確かめてください。

GSE1020E "<'spatial-reference-id>" is an invalid spatial reference system ID.

説明: "<'spatial-reference-id>" の識別子を持つ地理情報参照システムは使用可能になっていません。

ユーザーの応答: 指定した地理情報参照が使用可能になっていることを確かめてください。

GSE1021E Spatial Extender could not enable spatial reference system "<spatial-reference-id>" because the corresponding spatial coordinate system ID "<spatial-coordinate-id>" is invalid.

説明: "<spatial-coordinate-id>" の識別子を持つ座標系が、地理情報エクステンダー・カタログに定義されていません。

ユーザーの応答: 地理情報エクステンダー・カタログの DB2GSE.COORD_REF_SYS を調べて、座標系の識別子 "<spatial-coordinate-id>" を検査してください。

GSE1030E Because "<schema-name.table-name>" is not a base table, Spatial Extender could not enable a geocoder for it.

説明: ジオコーディングしたいソース・データを含むオブジェクトは基礎表でなければなりません。

ユーザーの応答: ジオコーディングしたいソース・データを含む列が基礎表の一部であることを確かめてください。

GSE1031E Spatial Extender could not enable geocoder "<geocoder-id>" to operate automatically in create mode for layer "<layer-schema.layer-name.layer-column>".

説明: 次のような説明が考えられます。

- ジオコーダーはレイヤー “<layer-schema.layer-name.layer-column>” を自動的に更新するためにすでに使用可能になっている。
- ジオコーダーはこのレイヤーに関しては一時的に無効になっている。
- このレイヤーには、ソース・データの列が定義されていない。

ユーザーの応答: ジオコーダーが一時的に無効になっている場合は、「再作成」モードで自動的に操作できるようにジオコーダーを使用可能にしてください。

GSE1032E Spatial Extender could not enable geocoder “<geocoder-id>” to operate automatically in recreate mode for layer “<layer-schema.layer-name.layer-column>”.

説明: 次のような説明が考えられます。

- ジオコーダーはレイヤー “<layer-schema.layer-name.layer-column>” を自動的に更新するためにすでに使用可能になっている。
- ジオコーダーはこのレイヤーについて無効になっていなかった。
- このレイヤーには、ソース・データの列が定義されていない。

ユーザーの応答: ジオコーダーがすでに除去モードで使用不可になっていた場合、あるいはこのレイヤー用に定義されていなかった場合、「作成」モードで自動的に操作できるようにジオコーダーを使用可能にしてください 。

GSE1033E An SQL error occurred when Spatial Extender tried to add triggers to a table that contains the column for layer “<layer-schema.layer-name.layer-column>” (SQLSTATE “<sqlstate>”).

説明: トリガーの目的は、ジオコーダーの入力元となる属性列と出力先となる地理情報列の間のデータ保全性を保つことです。 DB2 がこれらのトリガーを作成しようとしたときに、SQL エラーが起きました。

ユーザーの応答: SQLSTATE “<sqlstate>” に関連付けられたメッセージを検索してください。

GSE1034E Spatial Extender could not disable geocoder “<geocoder-id>” in drop mode for layer “<layer-schema.layer-name.layer-column>”.

説明: 次のような説明が考えられます。

- ジオコーダーはレイヤー “<layer-schema.layer-name.layer-column>” を自動的に更新できるように使用可能になっていなかった。
- ジオコーダーは除去モードで使用不可になっていた。

ユーザーの応答: ジオコーダーを使用不可にする前に、その状態を判別してください。たとえば、ジオコーダーは登録されていましたか。また、使用可能になっていましたか。その後、ジオコーダーを除去モードで使用不可にする必要があるかどうかを決定してください。たとえば、ジオコーダーが使用可能になっていなかった場合、それを使用不可にする必要は全くありません。

GSE1035E Spatial Extender could not disable geocoder “<geocoder-id>” in invalidate mode for layer “<layer-schema.layer-name.layer-column>”.

説明: 次のような説明が考えられます。

- ジオコーダーはレイヤー “<layer-schema.layer-name.layer-column>” を自動的に更新できるように使用可能になっていなかった。
- ジオコーダーは無効モードまたは除去モードで使用不可になっていた。

ユーザーの応答: ジオコーダーを使用不可にする前に、その状態を判別してください。たとえば、ジオコーダーは登録されていましたか。また、使用可能になっていましたか。その後、ジオコーダーを無効モードで使用不可にする必要があるかどうかを決定してください。たとえば、ジオコーダーがすでに無効モードで使用不可になっていた場合、それを再び同じモードで使用不可にする必要はありません。

GSE1036E An SQL error occurred when Spatial Extender tried to drop triggers from a table that contains the column for layer “<layer-schema.layer-name.layer-column>” (SQLSTATE “<sqlstate>”).

説明: トリガーは、ジオコーダーの入力元となる属性列と出力先となる地理情報列の間のデータ保全性を保つために作成されました。 DB2 がこれらのトリガーを除去しようとしたときに、SQL エラーが起きました。

ユーザーの応答: SQLSTATE “<sqlstate>” に関連付けられたメッセージを検索してください。

GSE1037E Spatial Extender could not geocode source data for table layer “<layer-schema.layer-name.layer-column>”, possibly because an incorrect value “<number-of-attributes>” was assigned to the argument that specifies how many attribute columns are to provide source data for this layer.

説明: このレイヤーに関連付けられた属性列に指定された数が間違っているか、それらの属性列の 1 つまたは複数の列に指定された名前が間違っています。

ユーザーの応答: そのレイヤーが関連属性列の正しい数と名前を使って登録されていることを確かめ、ジオコーダーの入力および出力データが正しいことを確認してください。

GSE1038E An SQL error occurred when Spatial Extender tried to geocode source data for table layer “<layer-schema.layer-name.layer-column>” in batch mode (SQLSTATE “<sqlstate>”).

ユーザーの応答:

- SQLSTATE “<sqlstate>” に関連付けられたメッセージを検索してください。
- このレイヤーの内容と primaryUDF 引き数の定義が正しいことを確かめてください。

GSE1050E The grid size that you specified (“<grid-size>”) is invalid for the first grid level.

説明: 最初の格子レベルの格子サイズにゼロまたは負の数を指定しました。

ユーザーの応答: 格子サイズに正の数を指定してください。

GSE1051E The grid size that you specified ("<grid-size>") is invalid for the second and third grid levels.

説明: 2 番目または 3 番目の格子レベルの格子サイズに負の数を指定しました。

ユーザーの応答: 格子サイズにゼロまたは正の数を指定してください。

GSE1052E An SQL error occurred when the Spatial Extender tried to create spatial index "<index-schema.index-column>" for a table layer "<layer-schema.layer-name.layer-column>" (SQLSTATE "<sqlstate>").

ユーザーの応答:

- 地理情報索引の指定が正確で、地理情報列に関連付けられた索引がないことを確かめてください。
- SQLSTATE "<sqlstate>" に関連付けられたメッセージを検索してください。

GSE1500I Source record "<record-number>" was successfully geocoded.

説明: 属性データが入っているレコードのジオコーディングが正常に行われました。

GSE1501W Source record "<record-number>" was not geocoded.

説明: 精度レベルが高過ぎます。

ユーザーの応答: さらに低い精度レベルでジオコーディングしてください。

GSE1502W Source record "<record-number>" was not found.

ユーザーの応答: データベースにレコードがあるかどうかを判別してください。

GSE2001E The specified transfer file ("<filename>") is not valid.

ユーザーの応答: 指定されたファイルが SDE 転送ファイルであり、パス名が正確に指定されていることを確かめてください。

GSE2002E The supplied SQL WHERE clause ("<SQL-where-clause>") is not valid.

ユーザーの応答: WHERE 文節を調べて、SQL 構文が適切か、つづりの誤りがないか、列名が無効でないかを確かめてください。

GSE2003E The supplied shape value is not legal.

ユーザーの応答: 提供された形状が、指定された地理情報列タイプと一致しているかどうかを確認してください。両方のタイプが一致しているか、互換性を持っている場合、図形の形式が不当です。重なり合うポリゴン、単一のポイントの弧などを調べてください。

GSE2004E The transfer file schema is incompatible with the schema of the specified layer.

ユーザーの応答: スキーマおよびレイヤー名の指定が適切であることを確認してください。スキーマが一致しなければ、データを新規の表としてロードし、スキーマの相違を解決してください。

GSE2005E The transfer file geometry type is incompatible with the geometry type of the specified layer.

ユーザーの応答: スキーマおよびレイヤー名の指定が適切であることを確認してください。

GSE2006E An I/O error for a file named "<filename>" has occurred.

ユーザーの応答: ファイルが存在すること、ファイルへのアクセスが適正であること、およびファイルが別のユーザーによって使用されていないことを確かめてください。

GSE2007E An attribute conversion error has occurred.

ユーザーの応答: 表内のすべての属性タイプがサポートされているかどうか調べてください。たとえば、BLOB データは形状ファイルでサポートされていません。また、範囲外のデータ値や、無効な日付などの不当なデータ値がないかどうかも調べてください。

GSE2008E The import/export function has run out of memory.

ユーザーの応答: 適切なメモリーが使用可能であることを確かめてください。

第11章 カタログ視点

DB2 地理情報エクステンダーのカタログ視点には、以下のものに関するメタデータが入っています。

- 使用できる座標系。これらのシステムの識別子および注釈テキストなどの情報については、 『DB2GSE.COORD_REF_SYS』を参照してください。
- レイヤーとして登録されている地理情報列。これらの列の名前、データ・タイプ、および関連する地理情報参照システムなどの情報については、 120ページの『DB2GSE.GEOMETRY_COLUMNS』を参照してください。
- 使用できるジオコーダー。これらのジオコーダーの識別子と、ジオコーダーが処理する場所を含む地域などの情報については、 120ページの『DB2GSE.SPATIAL_GEOCODER』を参照してください。
- 使用できる地理情報参照システム。地理情報参照システムの識別子とその記述などの情報については、 121ページの『DB2GSE.SPATIAL_REF_SYS』を参照してください。

DB2GSE.COORD_REF_SYS

地理情報操作のデータベースを使用可能にするとき、 DB2 地理情報エクステンダーは、カタログ表で使用できる座標系を登録します。この表から選択した列が、 DB2GSE.COORD_REF_SYS カタログ視点 (表34 に記述) を構成します。

表 34. DB2GSE.COORD_REF_SYS カタログ視点の列

名前	データ・タイプ	ヌル可能?	内容
SCID	INTEGER	いいえ	この座標系の固有の数値識別子。
SC_NAME	VARCHAR(64)	いいえ	この座標系の名前。
AUTH_NAME	VARCHAR(256)	はい	この座標系を編さんした母体組織の名前。たとえば、European Petroleum Survey Group (EPSG)。
AUTH_SRID	INTEGER	はい	AUTH_NAME 列に指定された組織によってこの座標系に割り当てられた数値識別子。
DESC	VARCHAR(256)	はい	この座標系の記述。
SRTEXT	VARCHAR(2048)	いいえ	この座標系の注釈テキスト。

DB2GSE.GEOMETRY_COLUMNS

レイヤーを作成するとき、 DB2 地理情報エクステンダーはレイヤーの識別子と関連情報をカタログ表に記録してレイヤーを登録します。この表から選択した列が、 DB2GSE.GEOMETRY_COLUMNS カタログ視点 (表35 に記述) を構成します。

表 35. DB2GSE.GEOMETRY_COLUMNS カタログ視点の列

名前	データ・タイプ	ヌル可能?	内容
LAYER_CATALOG	VARCHAR(30)	はい	このレイヤーの完全修飾名
LAYER_SCHEMA	VARCHAR(30)	いいえ	このレイヤーとして登録された列を含む表または視点のスキーマ。
LAYER_NAME	VARCHAR(128)	いいえ	このレイヤーとして登録された列を含む表または視点の名前。
LAYER_COLUMN	VARCHAR(30)	いいえ	このレイヤーとして登録された列の名前。
GEOMETRY_TYPE	INTEGER	いいえ	このレイヤーとして登録された列のデータ・タイプ。
SRID	INTEGER	いいえ	このレイヤーとして登録された列の値に使用する地理情報参照システムの識別子。
STORAGE_TYPE	INTEGER	はい	このレイヤーとして登録された列に DB2 が値を格納する方法についての情報。たとえば、 STORAGE_TYPE のデータでは、ラージ・オブジェクト (LOB) または抽象データ・タイプ (ADT) のインスタンスとして値を格納することを示します。

DB2GSE.SPATIAL_GEOCODER

使用可能なジオコーダーはカタログ表に登録されています。この表から選択した列が、 DB2GSE.SPATIAL_GEOCODER カタログ視点 (表36 に記述) を構成します。

表 36. DB2GSE.SPATIAL_GEOCODER カタログ視点の列

名前	データ・タイプ	ヌル可能?	内容
GCID	INTEGER	いいえ	このジオコーダーの数値識別子
GC_NAME	VARCHAR(64)	いいえ	このジオコーダーの短い説明。
VENDOR_NAME	VARCHAR(128)	いいえ	このジオコーダーを提供したベンダーの名前。
PRIMARY_UDF	VARCHAR(256)	いいえ	このジオコーダーの完全修飾名。

表 36. DB2GSE.SPATIAL_GEOCODER カタログ視点の列 (続き)

名前	データ・タイプ	ヌル可能?	内容
PRECISION_LEVEL	INTEGER	いいえ	ジオコーダーが正常な処理を行えるようにソース・データが対応参照データと一致しなければならない度合い。
VENDOR_SPECIFIC	VARCHAR(256)	はい	ベンダーがこのジオコーダーのサポートする任意の特殊パラメーターを設定するために使用できるファイルのパスと名前。
GEO_AREA	VARCHAR(256)	はい	ジオコーディングされる場所が含まれる図形領域。
DESCRIPTION	VARCHAR(256)	はい	ベンダーによって提供された注釈。

DB2GSE.SPATIAL_REF_SYS

地理情報参照システムを作成するとき、 DB2 地理情報エクステンダーはレイヤーの識別子と関連情報をカタログ表に記録してレイヤーを登録します。この表から選択した列が、 DB2GSE.SPATIAL_REF_SYS カタログ視点 (表37 に記述) を構成します。

表 37. DB2GSE.SPATIAL_REF_SYS カタログ視点の列

名前	データ・タイプ	ヌル可能?	内容
SRID	INTEGER	いいえ	この地理情報参照システムのユーザー定義識別子。
SR_NAME	VARCHAR(64)	いいえ	この地理情報参照システムの名前。
SCID	INTEGER	いいえ	この地理情報参照システムの基礎となる座標系の数値識別子。
SC_NAME	VARCHAR(64)	いいえ	この地理情報参照システムの基礎となる座標系の名前。
AUTH_NAME	VARCHAR(256)	はい	この地理情報参照システムの基準を設定する組織の名前。
AUTH_SRID	INTEGER	はい	AUTH_NAME 列で指定された組織がこの地理情報参照システムに割り当てる識別子。
SRTEXT	VARCHAR(2048)	いいえ	この地理情報参照システムの注釈テキスト。

第12章 地理情報索引

地理情報列には 2 次元の図形データが含まれているため、この列を照会するアプリケーションには、所定のエクステント内にあるすべての図形をすばやく識別する索引戦略が必要です。このため、DB2 地理情報エクステンダーには、格子に基づいた 3 層の地理情報索引が備わっています。

この章では、そのような索引について取り上げ、その使用方法に関する指針を提供します。以下のトピックを取り上げます。

サンプル・プログラムの断片

索引が作成されて SQL で使用される仕組みを示した次の例を考慮してください。 CREATE INDEX および CREATE INDEX EXTENSION コマンドの詳細については、 *SQL 解説書* を参照することができます。索引を作成した後は、地理情報の関数と述部を使用する標準的な DDL および DML ステートメントを発行できます。

```
create table customers (cid int, addr varchar(40), ..., loc db2gse.ST_Point)
create table stores (sid int, addr varchar(40), ..., loc db2gse.ST_Point,
 zone db2gse.ST_Polygon)
create index customersx1 on customers(loc) extend using spatial_index(10e0, 100e0,
 1000e0)
create index storesx1 on stores(loc) extend using spatial_index(10e0, 100e0,
 1000e0)
create index storesx2 on stores(zone) extend using spatial_index(10e0, 100e0,
 1000e0)

insert into customers (cid, addr, loc) values (:cid, :addr, sdeFromBinary(:loc))
insert into customers (cid, addr, loc) values (:cid, :addr, geocode(:addr))
insert into stores (sid, addr, loc) values (:sid, :addr, sdeFromBinary(:loc))

update stores set zone = db2gse.ST_Buffer (loc, 2)

select cid, loc from customers
 where db2gse.ST_Within(loc, :polygon) = 1

select cid, loc from customers
 where db2gse.ST_Within(loc, :circle1) = 1 OR
       db2gse.ST_Within(loc, :circle2) = 1
```

```
select c.cid, loc from customers c, stores s
 where db2gse.ST_Contains(s.zone, c.loc) = 1 selectivity 0.01

select avg(c.income) from customers c
 where not exist (select * from stores s
                  where db2gse.ST_Distance(c.loc, s.loc) < 10)
```

B ツリー索引

地理情報索引作成テクノロジーは、従来の階層式 B ツリー索引に準拠していますが、大幅に異なる点があります。地理情報索引は、2 次元の地理情報列の索引を作成するために設計された*格子索引作成* を使用します。 B ツリー索引が扱えるのは 1 次元のデータだけで、GIS 情報で使用することはできません。ここでは、B ツリー索引を構造化して使用する方法について説明します。

B ツリー索引の最上レベル (ルート・ノードという) には、1 つ下位レベルのノードごとに、 1 つのキーが含まれています。各キーの値は、1 つ下位レベルにある対応するノードの既存キー値のうち、最大のものです。基礎表にある値の数によっては、複数の中間ノードが必要になることがあります。これらのノードは、ルート・ノードと、実際の基礎表の行 ID を保持するリーフ・ノードとの間のブリッジとなります。

データベース・マネージャーは、ルート・ノードから始まる B ツリー索引を検索します。その後、基本表の行 ID を持つリーフ・ノードに到達するまで、中間ノードを検索してゆきます。

2 次元という特性を持つ地理情報列には、地理情報索引の構造が必要であるため、 B ツリー索引を地理情報列に適用することはできません。同じ理由で、地理情報索引を非地理情報列に適用することもできません。さらに、地理情報索引は、どの種類の複合列にも適用することはできません。

地理情報索引を作成する方法

地理情報索引を作成する場合、次のいくつかの方法があります。

- 「地理情報索引の作成 (Create Spatial Index)」ウィンドウから定義する。説明については、55ページの『第6章 地理情報索引の作成』を参照してください。
- アプリケーション・プログラムで db2gse.gse_enable_idx ストアード・プロシージャーを呼び出す。このストアード・プロシージャーについての詳細は、71ページの『第9章 ストアード・プロシージャー』を参照してください。
- USING 文節に **spatial_index** 関数を指定した **db2 create index** コマンドを発行する。たとえば次のようにします。

```
create index storesx1 on customers (loc) using spatial_index(10e0, 100e0, 1000e0)
```

地理情報データの性質上、データベース設計者は相対的なサイズの分散について理解しておく必要があります。そして設計者は、最適サイズと、地理情報索引を作成するときの格子レベルの数を決定しなければなりません。

格子レベルの <grid level 1>、<grid level 2>、および <grid level 3> は、セルのサイズを大きくすることにより指定されます。したがって、2 番目のレベルのセル・サイズは最初のものよりも大きく、 3 番目のレベルのセル・サイズは 2 番目のものよりも大きくなります。最初の格子レベルは必須ですが、 2 番目と 3 番目の格子レベルについては倍精度のゼロ値 (0.0e0) を指定して使用不可にすることができます。

地理情報索引が生成される方法

地理情報索引は、エンベロープ を使用して生成されます。エンベロープは、図形そのもので、特定の図形の X と Y の長さのうち最小のものと最大のものを表します。ほとんどの図形では、エンベロープは箱型です。ただし水平折れ線と垂直折れ線の場合、エンベロープは 2 つのポイント間の折れ線になります。ポイントの場合、エンベロープはポイントそのものになります。エンベロープについての詳細は、137ページの『エンベロープ』を参照してください。

格子を利用して各図形のエンベロープの交差部分について 1 つまたは複数の項目を作成することにより、地理情報索引が地理情報列に対して作成されます。交差部分は、図形の内部 ID として、また交差部分となる格子セルの X および Y 座標の最小値として記録されます。たとえば、126ページの図7 のポリゴンは、座標 (20,30)、(30,30)、(40,30)、(20,40)、(30,40)、(40,40)、(20,50)、(30,50)、(40,50) の格子と交差します。 126ページの図7 の全図形の X および Y 座標の最小値については、 126ページの表38 を参照してください。

複数の格子レベルが存在する場合、 DB2 地理情報エクステンダーは可能な限り低い格子レベルを使おうとします。所定のレベルにおいて、1 つの図形が 4 つ以上の格子セルと交差する場合は、レベルが次に大きいレベルに昇格します。したがって、10.0e0、100.0e0、そして 1000.0e0 の 3 つの格子レベルのある地理情報索引の場合、 DB2 地理情報エクステンダーはまず、レベル 10.0e0 の格子と各図形を交差させます。指定した図形が 4 つ以上の 10.0e0 格子セルと交差している場合、昇格してレベル 100.0e0 の格子で交差するようになります。 4 つ以上の交差が 100.0e0 レベルで生じていると、この図形は 1000.0e0 レベルへ昇格します。これが一番高いレベルになるので、交差部分は 1000.0e0 レベルで地理情報索引に入力されなければなりません。

図7 では、 4 つの異なるタイプの図形が 10.0e の格子と交差しています。地理情報索引には、4 つの図形の 23 の交差部分すべてが記録されます。

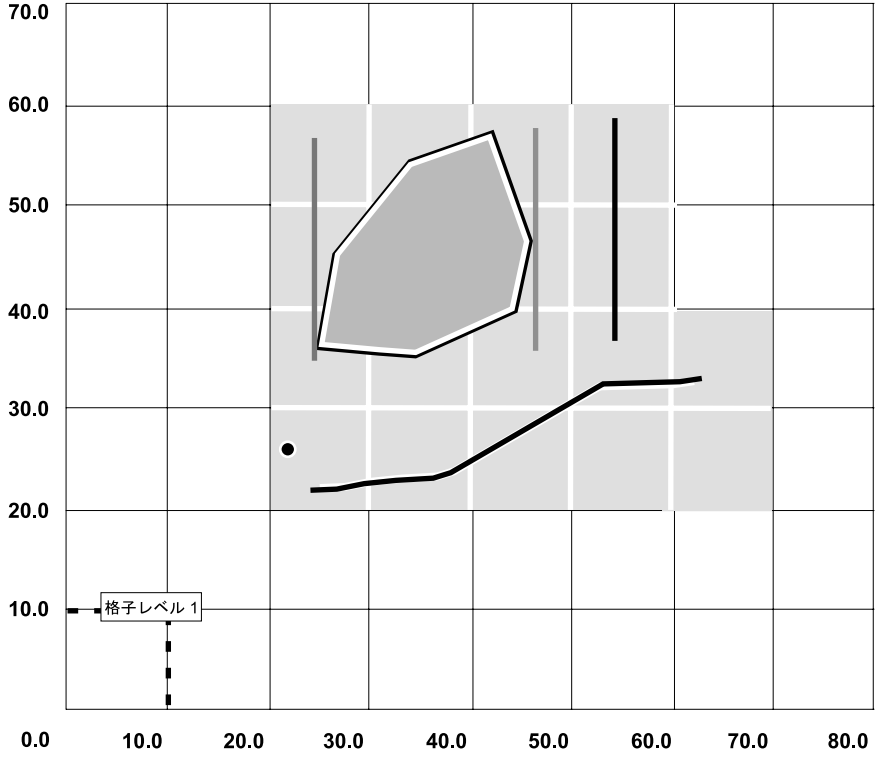


図 7. 10.0e0 格子レベルの適用

表38 には、図形とそれに対応する格子の交差部分がリストされています。 4 つの異なるタイプの図形のエンベロープが、10.0e の格子と交差しています。地理情報索引には、交差されている各格子セルの最小 X 座標および Y 座標が入力されます。

表 38. 図形例の 10.0e0 格子セルの項目

図形	格子 X	格子 Y
ポリゴン	20.0	30.0
ポリゴン	30.0	30.0
ポリゴン	40.0	30.0
ポリゴン	20.0	40.0
ポリゴン	30.0	40.0

表 38. 図形例の 10.0e0 格子セルの項目 (続き)

図形	格子 X	格子 Y
ポリゴン	40.0	40.0
ポリゴン	20.0	50.0
ポリゴン	30.0	50.0
ポリゴン	40.0	50.0
垂直折れ線	50.0	30.0
垂直折れ線	50.0	40.0
垂直折れ線	50.0	50.0
ポイント	20.0	20.0
水平折れ線	20.0	20.0
水平折れ線	30.0	20.0
水平折れ線	40.0	20.0
水平折れ線	50.0	20.0
水平折れ線	60.0	20.0
水平折れ線	20.0	30.0
水平折れ線	30.0	30.0
水平折れ線	40.0	30.0
水平折れ線	50.0	30.0
水平折れ線	60.0	30.0

128ページの図8 には、格子レベル 30.0e0 および 60.0e0 を追加することにより交差部分の数が 8 まで激減した様子が示されています。この場合、図形 1 のポリゴンは格子レベル 30.0e0 に昇格し、図形 4 の折れ線は格子レベル 60.0e0 へ昇格します。 10.0e0 レベルでは、その図形には 9 と 10 の交差部分がありましたが、昇格後は交差部分は 2 つだけになります。

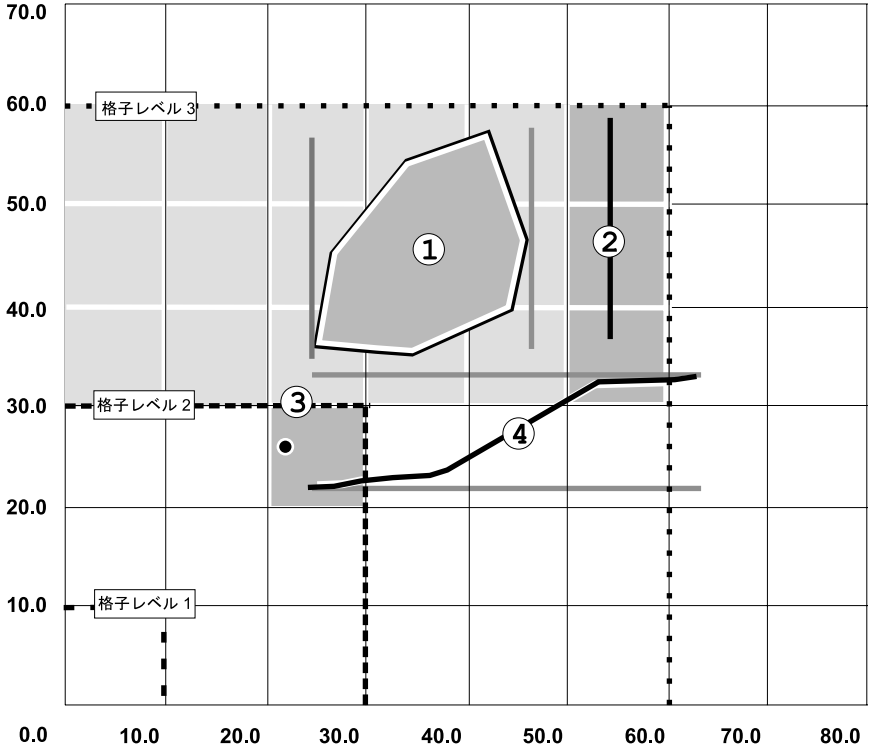


図 8. 格子レベル 30.0e0 と 60.0e0 を追加したときの結果. 図形 1 の多角形のエンベロープは、9 つの格子セルと交差しています。図形 2 の垂直折れ線のエンベロープは、3 つの格子セルと交差しています。図形 3 のポイントのエンベロープが交差しているのは、1 つの格子セルだけです。図形 4 の折れ線のエンベロープは、10 の格子セルと交差しています。

DB2 地理情報エクステンダーは CREATE INDEX ステートメントで指定された格子レベル・パラメーターを取り、個々の地理情報オブジェクトを調べて、オブジェクトが存在する格子ブロックの座標と番号を判別します。 図8 の場合、格子レベル 10.0e0、30.0e0、および 60.0e0 では、線が太くなっていたり、グレーの濃さが異なっていたりしています。垂直折れ線とポイント・エンベロープ・セルとの交差部分は 4 つ未満なので、 10.0e0 格子レベルの索引に入力されます。ポリゴンは 9 つの 10.0e0 格子セルと交差するので、30.0e0 格子レベルへ昇格します。このレベルでは、ポリゴンは 2 つの格子セルと交差しており、その交差部分が索引に入力されます。図形 4 の折れ線は 10 の 10.0e0 格子セルと交差するので、30.0e0 格子レベルへ昇格します。このレベルでも 6 つの格子セルと交差するため、60.0e0 格子レベルへ昇格します。ここでは交差部分が 2 つとなります。したがって、折れ線 60.0e0 の格子との交差部分が索引に入力されます。このレベルで折れ線との交差部分が 4 つ以上となっても、図形を昇格できる最高のレベルであるため、その交差部分が索引に入力されま

す。

表 39. 3 層索引での図形の交差部分

図形	格子 X	格子 Y
レベル 1 (10.0e0 格子サイズ) での垂直折れ線とポイントとの交差部分		
2	50.0	30.0
2	50.0	40.0
2	50.0	50.0
3	20.0	20.0
レベル 2 (30.0e0 格子サイズ) でのポリゴンの交差部分		
1	0.0	30.0
1	30.0	30.0
レベル 3 (60.0e0 格子サイズ) での折れ線の交差部分		
4	0.0	0.0
4	60.0	0.0

DB2 地理情報エクステンダーは、実際にはどのような種類のポリゴン格子構造も作成しません。さらに DB2 地理情報エクステンダーは、列の地理情報参照システムの X、Y オフセットで原点を定義することにより、それぞれの格子レベルをパラメーターで示します。その後、格子を正の座標空間に展開します。パラメトリック格子を使用することにより、DB2 地理情報エクステンダーは交差部分を数学的に生成します。

地理情報索引の使用に関する指針

DB2 地理情報エクステンダーは地理情報索引と連動し、地理情報の照会のパフォーマンスを向上させます。最も基本的かつ最も一般的な地理情報の照会、つまりボックス照会を考えてみましょう。この照会では、ユーザー定義のボックス内に全体あるいは一部が含まれる図形をすべて戻すよう DB2 地理情報エクステンダーに問い合わせます。索引が存在しない場合、DB2 地理情報エクステンダーはボックス内のすべての図形を比較しなければなりません。しかし索引があれば、DB2 地理情報エクステンダーは、左下の座標がボックスの座標と等しいかそれよりも大きいすべての索引項目、および右上の座標がボックスの座標と等しいかそれよりも小さいすべての索引項目を見付けることができます。索引はこの座標系によって順序付けされているので、 DB2 地理情報エクステンダーは候補となる図形のリストをすぐに入手することができます。ここで説明した処理のことを、*第 1 パス* といいます。

第 2 パス では、各候補のエンベロープがボックスと交差しているかどうかを判別します。格子セルのエンベロープがボックスと交差しているために第 1 パスで入手された図形には、交差しないエンベロープが含まれていることがあります。

第 3 パス では、候補の実際の座標をボックスと比較して、図形のいずれかの部分が実際にボックス内に含まれているかどうかを判別します。この最後に行うやや複雑な比較処理は、全体のサブセット (最初の 2 つのパスでかなり減少している) で構成されている候補のリストに対して行われます。

EnvelopesIntersect 関数を除き、すべての地理情報照会ではこの 3 つのパスを実行します。この関数は最初の 2 つのパスだけを実行します。
EnvelopesIntersect 関数は、表示操作用に設計されています。表示操作は多くの場合、独自の組み込みクリッピング・ルーチンを採用しており、第 3 パスによるきめ細かさを必要としないからです。

格子セル・サイズの選択

図形エンベロープの形状が不規則であると、格子セル・サイズの選択は複雑になります。不規則な形状のため、複数の格子と交差する図形エンベロープや、1 つの格子セル内に収まる図形エンベロープが出てくることがあります。逆に、地理情報データの分散状況によっては、多数の図形エンベロープと交差する格子セルが出てくることもあります。

地理情報索引をうまく機能させるには、正しい数およびサイズの格子を選択することが重要です。サイズが統一されている図形を含む地理情報列を考慮してみましょう。この場合、1 つの格子レベルで十分です。平均的な図形エンベロープを含めることのできる格子セル・サイズから開始します。アプリケーションをテストしているときに、格子セル・サイズを大きくすると照会のパフォーマンスが向上することに気付きます。これは、各格子セルにさらに多くの図形が含まれるようになり、第 1 パスが入手されていない図形をより速く廃棄できることによります。しかし、セル・サイズを大きくし続けると、パフォーマンスが悪くなってゆくことも分かります。結局は第 2 パスでさらに多くの候補を処理しなければならなくなるからです。

レベル数の選択

索引作成の対象となるオブジェクトの相対サイズが同じであれば、単一の格子レベルを使用することができます。たとえそうではあっても、すべての列に同じ相対サイズの図形が含まれているわけではありません。地理情報列の図形は普通、いくつかのサイズごとにグループ化することができます。たとえば、図形が一般道、幹線道路および高速道路に分かれている道路網を考えます。道路

はどれもみな同じ長さで、1 つのサイズ間隔でグループ化できます。これは、幹線道路と高速道路にも当てはまります。したがって、1 つのサイズ間隔を表す道路は第 1 格子レベルに、道路網は第 2 格子レベル、幹線高速道路は第 3 格子レベルにグループ化できます。別の例として、小さな都市区画の集まりが含まれ、その区画がより大きな郊外区画に囲まれている郡区画列もあります。このインスタンスでは、2 つのサイズ間隔と 2 つの格子レベルがあり、一方は小さい都市区画用、他方はより大きな郊外区画用です。このような状況は非常に一般的であり、複数レベルの格子を使用する必要があります。

各格子レベルのセル・サイズを選択するには、各サイズ間隔より少し大きめの格子セル・サイズを選択してください。地理情報列に対して照会を実行することにより、索引をテストしてください。

レベルを追加すると、索引を余分に走査しなければなりません。格子のサイズを微調整し、パフォーマンスが目に見えて向上するかどうかを判別してください。

第13章 図形および関連する地理情報関数

この章では、座標を構成し、地形を記号化する*図形* と呼ばれる情報の単位について説明します。また、図形を入力データとして処理する地理情報関数についても紹介します。これらの関数は、地形を分析し、図形情報システム間で地理情報データを移動するのに役立つ結果を戻します。以下のトピックを取り上げます。

- 図形の性質
- 図形の特性、およびこれらの特性に関連する情報を戻す関数
- インスタンス化可能な図形、およびそれらの図形を操作する関数
- 以下のことを行う関数。
  - 地形間の関係および比較の表示
  - 図形の生成
  - インポートおよびエクスポート可能な形式への図形値の変換

図形について

Oxford American Dictionary は、*geometry (幾何学)* を「the branch of mathematics dealing with the properties of and relations of lines, angles, surfaces and solids (線、角度、面、および立体の特性と関係を扱う数学の一分野)」と定義しています。 1997 年 8 月 11 日 に、Open GIS Consortium Inc. (OGC) は、 *Open GIS Features for ODBC (SQL) Implementation Specification* という出版物で、この用語に対して別の定義を行いました。地球上の地図を作成するために、過去 1000 年以上の間地図作製者によって使用されてきた幾何学的な地形を表すため、 *geometry (図形)* という語が選ばれました。 geometry (図形、または幾何学) というこの新しい意味の定義は、「地形を記号化するポイントまたはポイントの集合」というように要約できるかもしれません。

DB2 地理情報エクステンダーでは、図形は*操作に関して*、「地形のモデル」と定義することができます。モデルは、地形の座標を示す用語で、また場合によっては可視記号を示す用語で表すことができます。モデルは、情報を伝えます。たとえば、座標は固定参照ポイントに関する地形の位置を識別し、記号はその形を略図として示します。また、モデルを使って情報を生成することもできます。たとえば、 ST_Overlaps 関数は 2 つの最も近い領域の座標を入力データとして受け取り、それらの領域の重なり合いを示す情報を戻します。

図形が記号化する地形の座標は、図形の*特性* とみなされます。いくつかの種類の図形は、その他の特性を持っています。以下に例を示します。

- *内部* は、図形が記号化する地形の内容を表します。
- *外部* は、地形を取り囲むスペースを表します。
- *境界* は、内容が終わり、周囲のスペースが始まる境界を表します。

これらの特性と追加の特性については、135ページの『図形の特性および関連する関数』で説明されています。

DB2 地理情報エクステンダーでサポートされる図形は階層を形成します (図9 に表示)。 6 つの階層メンバーは、図にも示されているとおり、インスタンス化可能で、可視記号として表現できます。

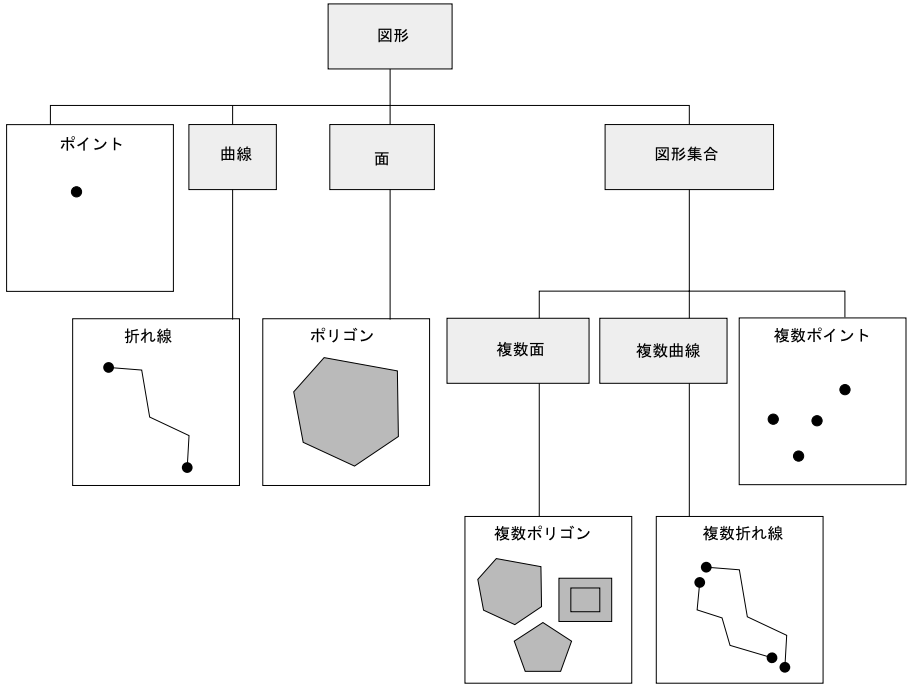


図 9. DB2 地理情報エクステンダーでサポートされる図形の階層. インスタンス化可能な図形は可視記号として表現できます。これらの記号は、該当する図形の名前の下に表示されます。

図9 が示すとおり、*図形* と呼ばれるスーパークラスは階層のルートです。サブクラスは、基本図形サブクラスと同種集合サブクラスという 2 つの区分に分けることができます。基本図形には、以下のものが含まれます。

- *ポイント*。東西の座標線 (緯線) が南北の座標線 (経線) と交差する場所を占めるものとして認識される特定の地形を記号化します。たとえば、大縮尺の

地図表記で、地図上の各都市が緯線と経線の交差部分に位置している場合を想定します。そのような縮尺では、各都市はポイントとして記号化できます。

- *折れ線*。線形の地勢 (たとえば、道路、運河、配管) を記号化します。
- *ポリゴン*。多角形の地勢 (たとえば、福祉地域、森林、野生生物の生息地) を記号化します。

同種集合には、次のものが含まれます。

- *複数ポイント*。東西座標線と南北座標線の交差部分に個々の構成要素が位置する複数パートの地形 (たとえば、個々のメンバーが緯線と経線の交差部分に位置する列島) を記号化します。
- *複数折れ線*。線形の構成要素単位で構成される複数パートの地形 (たとえば、河川システムや高速道路システム) を記号化します。
- *複数ポリゴン*。多角形の構成要素単位で構成される複数パートの地形 (たとえば、特定地域の集合農地や湖沼システム) を記号化します。

名前が示しているとおり、同種集合は基本図形の集合です。加えて、基本図形特性を共有するには、同種集合にさらに独自の特性がいくつかなければなりません。

DB2 地理情報エクステンダーによってサポートされる地理情報データ・タイプは、 134ページの図9 に示されている図形の実装です。これらのデータ・タイプの説明については、35ページの『地理情報データ・タイプについて』を参照してください。

図形の特性および関連する関数

ここでは、図形の特性と、それらの特性に関連する地理情報関数について説明します。まず、中核を成す特性から始めます。

- 図形が属するクラス
- X および Y 座標

ここでは、次の点も説明します。

- Z 座標
- 測定値
- 図形の内部、境界、および外部
- 単純または非単純という品質
- 空または空でないという品質
- 図形のエンベロープ

- 次元
- 図形の関連地理情報参照システムの識別子

クラス

各図形は、134ページの図9 に示されている階層内のクラスに属しています。133ページの『図形について』 に示されているとおり、階層内の 6 つのサブクラス、すなわち、ポイント、折れ線、ポリゴン、複数ポイント、複数折れ線、および複数ポリゴンはインスタンス化可能です。スーパークラスとその他のサブクラスはインスタンス化可能ではありません。

ST_GeometryType 関数は、図形を受け取り、文字ストリングの形式でインスタンス化可能サブクラスを戻します。詳細については、222ページの『ST_GeometryType』を参照してください。

ST_IsValid 関数は、ST_Geometry データ・タイプに割り当てられた図形を受け取ります。この関数は、図形が有効であれば 1 (TRUE) を戻し、図形が無効であれば 0 (FALSE) を戻します。詳細については、238ページの『ST_IsValid』を参照してください。

X および Y 座標

X 座標値 は、東西の参照ポイントからの相対位置を表します。 *Y 座標値* は、北南の参照ポイントからの相対位置を表します。詳細については、5ページの『地理情報データの性質』および 27ページの『座標系および地理情報参照システムについて』を参照してください。

Z 座標

図形の中には、高さや深さが関係するものもあります。地形の図形を形成する各ポイントには、任意指定の Z 座標があることがあります。 Z 座標は地表面を基準とした高さや深さを表します。

Is3d 述部関数は、図形を受け取り、関数が Z 座標を持っていれば 1 (TRUE)、それ以外の場合は 0 (FALSE) を戻します。詳細については、173ページの『Is3d』を参照してください。

測定値

測定値は、地形に関する情報を伝え、地形の場所を定義する座標と一緒に格納される値です。たとえば、GIS で交通システムを表す場合を想定します。アプリケーションで線形距離や里程標を表す値を処理したい場合は、システムの位置を定義する座標と一緒にこれらの値を格納することができます。測定値は、倍精度の数値で格納されます。

IsMeasured 述部関数は、図形を受け取り、それが測定値を含んでいる場合は 1 (TRUE)、それ以外の場合は 0 (FALSE) を戻します。詳細については、174ページの『IsMeasured』を参照してください。

内部、境界、外部

図形はすべて、内部、境界、および外部により定義されているスペース内の位置を占めています。図形の外部とは、図形が占めていないスペース全体のことです。図形の境界は、内部と外部の間の接面の役割を果たします。内部とは、図形が占めているスペースのことです。サブクラスは、内部および外部特性を直接継承しますが、境界特性はそれぞれ異なります。

ST_Boundary 関数は、図形を受け取り、ソース図形の境界を表す図形を戻します。詳細については、193ページの『ST_Boundary』を参照してください。

単純または非単純

いくつかの図形のサブクラス (折れ線、複数ポイント、複数折れ線) は、単純または非単純のいずれかになります。サブクラスに課せられている位相規則すべてに従っているサブクラスは単純で、そうでないサブクラスは非単純です。折れ線は、内部で交差していなければ単純です。複数ポイントは、同じ座標空間を占めている要素がなければ単純です。複数折れ線は、それ自体の内部と要素の内部が交差していなければ単純です。

ST_IsSimple 述部関数は図形を受け取り、その図形が単純であれば 1 (TRUE) を戻し、それ以外の場合は 0 (FALSE) を戻します。詳細については、237ページの『ST_IsSimple』を参照してください。

空または空でない

図形は、全くポイントがなければ空の図形です。空の図形のエンベロープ、境界、内部、および外部は NULL です。空の図形は常に単純であり、Z 座標または測定値を持つ場合があります。空の折れ線および複数折れ線の長さは 0 です。空のポリゴンおよび複数ポリゴンの面積は 0 です。

ST_IsEmpty 述部関数は図形を受け取り、その図形が空であれば 1 (TRUE) を戻し、それ以外の場合は 0 (FALSE) を戻します。詳細については、233ページの『ST_IsEmpty』を参照してください。

エンベロープ

図形のエンベロープとは、最小および最大の (X,Y) 座標によって形成されている、境界を作る図形のことです。以下の例外がありますが、ほとんどの図形のエンベロープは境界長方形を形成します。

- ポイントのエンベロープはそのポイント自体です。これはそのポイントの最小および最大座標が同じであるためです。
- 水平および垂直の折れ線のエンベロープは、ソースの折れ線の境界 (端点) によって表される折れ線です。

ST_Envelope 関数は、図形を受け取り、境界を作る図形 (エンベロープを表す) を戻します。詳細については、213ページの『ST_Envelope』を参照してください。

次元

図形には、0、1、または 2 の次元があります。それらの次元を以下にリストします。

0 長さも面積もない

1 長さがある

2 面積がある

ポイントおよび複数ポイントのサブクラスの次元は、ゼロです。ポイントは単純座標でモデル化できる次元の形を表し、複数ポイント・サブクラスは切り離された座標のクラスターでモデル化しなければならないデータを表します。

折れ線および複数折れ線のサブクラスの次元は 1 です。これらのサブクラスには、道路の一区切り、分水システム、および実在する線状の他の地形が格納されます。

ポリゴンおよび複数ポリゴンのサブクラスの次元は、2 です。森林、区画、水域など、周辺が定義可能な領域で囲まれている地形は、ポリゴンまたは複数ポリゴンのデータ・タイプで表すことができます。

次元はサブクラスの特性として重要なだけでなく、 2 つの地形の地理関係を判別する役割も果たします。結果の地形 (単数または複数) の次元によって、操作が成功したかどうかが判別されます。 DB2 地理情報エクステンダーは、地形の次元を調べ、それらを比較する方法を判別します。

ST_Dimension 関数は図形を受け取り、その次元を整数として戻します。詳細については、207ページの『ST_Dimension』を参照してください。

地理情報参照システムの識別子

地理情報参照システムは、図形ごとの座標変換を識別します。

データベースに認識されている地理情報参照システムはすべて、DB2GSE.SPATIAL_REF_SYS カタログ視点を使ってアクセスできます。この視点についての詳細は、119ページの『第11章 カタログ視点』を参照してください。

ST_SRID 関数は図形を受け取り、地理情報参照 ID を整数として戻します。詳細については、268ページの『ST_SRID』を参照してください。

ST_Transform 関数は、図形が現在割り当てられている地理情報参照システム以外の地理情報参照システムに、図形を割り当てます。詳細については、273ページの『ST_Transform』を参照してください。

インスタンス化可能な図形および関連する関数

ここでは、6 つのインスタンス化可能な図形について取り上げ、その操作に使用する関数を説明します。サブクラスは、次のとおりです。

- ポイント
- 折れ線
- ポリゴン
- 複数ポイント
- 複数折れ線
- 複数ポリゴン

これらのサブクラスが属する階層の図、および関連する可視記号については、134ページの図9 を参照してください。

ポイント

ポイントは、座標空間中の 1 つの位置を占める 0 次元の図形です。ポイントには、その位置を定義する X 座標と Y 座標が含まれます。また、Z 座標と測定値が含まれる場合もあります。

ポイントは単純で、NULL 境界を持っています。ポイントは、しばしば油井、陸標、高台などの地形を定義するのに使用されます。

以下の関数は、ポイント・サブクラスだけを操作するものです。

ST_Point

X 座標、関連する Y 座標、およびそれらの座標が属する地理情報参照システムの識別子を受け取り、座標が定義するポイントを戻します。詳細については、259ページの『ST_Point』を参照してください。

ST_CoordDim

ポイントに含まれる座標の数値と、それに測定値も含まれるかどうかを表す値を戻します。この値は座標の次元と呼ばれています。定義可能な座標の次元は以下のとおりです。

2 ポイントは X 座標と Y 座標で構成されます。

3 ポイントは、X 座標、Y 座標、および Z 座標で構成されます。

4 ポイントは、X 座標、Y 座標、Z 座標、および測定値で構成されます。

詳細については、202ページの『ST_CoordDim』を参照してください。

ST_PointFromText

ポイントの OGC 事前割り当てテキスト (WKB) 表現を受け取り、ポイントを戻します。詳細については、257ページの『ST_PointFromText』を参照してください。

ST_X 倍精度数として ST_Point ポイント・データ・タイプの X 座標値を戻します。詳細については、280ページの『ST_X』を参照してください。

ST_Y 倍精度数として ST_Point ポイント・データ・タイプの Y 座標値を戻します。詳細については、281ページの『ST_Y』を参照してください。

Z 倍精度数として ST_Point ポイント・データ・タイプの Z 座標値を戻します。詳細については、282ページの『Z』を参照してください。

M 倍精度数として ST_Point ポイント・データ・タイプの測定値を戻します。詳細については、181ページの『M』を参照してください。

折れ線

折れ線とは、一続きのポイントとして保管され、線状に補間された道を定義する 1 次元オブジェクトです。折れ線は、内部で交差していなければ単純です。閉じている折れ線では、複数の端点 (境界) がスペース中の同じポイントを占めています。折れ線が閉じていて、その内部自体が交差していなければ、その折れ線はリングです。スーパークラス図形から継承される他の特性に加えて、折れ線には長さがあります。折れ線はしばしば、道路、川、送電線などの線状の地形を定義するのに使用されます。

始点と端点が同じ単純な折れ線はリング と呼ばれます。

折れ線が閉じている (この場合、境界は NULL) のでない限り、通常は端点が折れ線の境界を形成します。折れ線の内部は、折れ線が閉じている (この場合、内部は連続) のでない限り、端点と端点をつなぐ道です。

折れ線を操作する関数を以下に示します。

ST_StartPoint

この関数は、折れ線を受け取り、最初のポイントを戻します。詳細については、269ページの『ST_StartPoint』を参照してください。

ST_EndPoint

この関数は、折れ線を受け取り、最後のポイントを戻します。詳細については、212ページの『ST_Endpoint』を参照してください。

ST_PointN

この関数は、折れ線と *n* 番目のポイントへの指数を受け取り、そのポイントを戻します。詳細については、260ページの『ST_PointN』を参照してください。

ST_Length

この関数は、折れ線を受け取り、倍精度数として長さを戻します。詳細については、240ページの『ST_Length』を参照してください。

ST_NumPoints

この関数は、折れ線を受け取り、つなげられているポイントの数を整数として戻します。詳細については、252ページの『ST_NumPoints』を参照してください。

ST_IsRing

この関数は、折れ線を受け取り、その折れ線がリングの場合は 1 (TRUE)、それ以外の場合は 0 (FALSE) を戻します。詳細については、235ページの『ST_IsRing』を参照してください。

ST_IsClosed

この関数は、折れ線を受け取り、その折れ線が閉じている場合は 1 (TRUE)、それ以外の場合は 0 (FALSE) を戻します。詳細については、231ページの『ST_IsClosed』を参照してください。

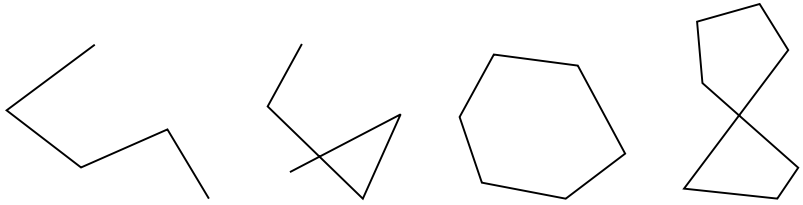


図 10. 折れ線オブジェクト.

1. 単純で閉じていない折れ線。
2. 非単純で閉じていない折れ線。
3. 閉じている単純な折れ線 (リング)。
4. 閉じている単純でない折れ線 (リングではない)。

ポリゴン

ポリゴンとは、一続きのポイントとして保管され、外部の境界を作るリングと 0 個以上の内部のリングを定義する、2 次元の面です。ポリゴンのリングは重なり合うことができません。したがって、ポリゴンは定義により、常に単純です。ほとんどの場合、土地の区画、水の集まっている場所、および地理情報的な範囲を持つ他の地形を定義します。

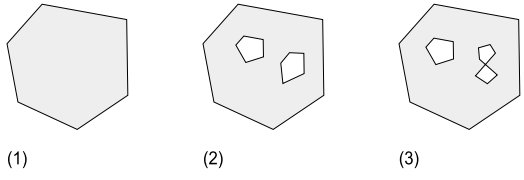


図 11. ポリゴン.

1. 境界が外部のリングで定義されているポリゴン。
2. 境界が 1 つの外部のリングと 2 つの内部のリングで定義されているポリゴン。内部のリングの内側の領域は、多角形の外部の一部です。
3. 正しいポリゴン (リングが 1 つの接点で交わっているため)。

外部および内部のリングはポリゴンの境界を定義し、リングの間で閉じられているスペースはポリゴンの内部を定義します。ポリゴンのリングは接点で交わることができますが、交差させることはできません。スーパークラス図形から継承される他の特性に加えて、ポリゴンには面積があります。

ポリゴンを操作する関数を以下に示します。

ST_Area

この関数は、ポリゴンを受け取り、倍精度数として面積を戻します。詳細については、189ページの『ST_Area』を参照してください。

ST_ExteriorRing

この関数は、ポリゴンを受け取り、折れ線として外部リングを戻します。詳細については、216ページの『ST_ExteriorRing』を参照してください。

ST_NumInteriorRing

この関数は、ポリゴンを受け取り、その多角形にある内部リングの数を戻します。詳細については、251ページの『ST_NumInteriorRing』を参照してください。

ST_InteriorRingN

この関数は、ポリゴンと指標を受け取り、折れ線として *n* 番目の内部リングを戻します。詳細については、224ページの『ST_InteriorRingN』を参照してください。

ST_Centroid

この関数は、ポリゴンを受け取り、ポリゴンの範囲の中央にあるポリゴンを戻します。詳細については、197ページの『ST_Centroid』を参照してください。

ST_PointOnSurface

この関数は、ポリゴンを受け取り、ポリゴンの面上にあることが保証されているポイントを戻します。詳細については、261ページの『ST_PointOnSurface』を参照してください。

ST_Perimeter

ポリゴンを受け取り、その面の周囲の長さを戻します。詳細については、256ページの『ST_Perimeter』を参照してください。

複数ポイント

複数ポイントはポイントの集合で、その要素と同様、0 次元です。複数ポイントは、同じ座標空間を占めている要素がなければ単純です。複数ポイントの境界は NULL です。複数ポイントは、放送アンテナの配置図や伝染病の発生地点などの事象を定義するために使用できます。

複数ポイントを操作する関数を以下に示します。

ST_NumGeometries

同種集合を受け取り、その中に含まれている基本図形要素の数を戻します。詳細については、250ページの『ST_NumGeometries』を参照してください。

ST_GeometryN

同種集合および指数を受け取り、何番目か (任意) の基本図形を戻します。詳細については、221ページの『ST_GeometryN』を参照してください。

複数折れ線

複数折れ線は、折れ線の集合です。複数折れ線は、折れ線要素の端点で交差している場合のみ、単純です。折れ線要素の内部が交差している場合、複数折れ線は単純ではありません。

複数折れ線の境界は、折れ線要素の交差していない端点です。複数折れ線が閉じているのは、折れ線要素がすべて閉じている場合です。すべての要素の端点すべてが交差している場合、複数折れ線の境界は NULL です。スーパークラス図形から継承される他の特性に加えて、複数折れ線には長さがあります。複数折れ線は、川の細流や道路網を定義するのに使用します。

複数折れ線を操作する関数を以下に示します。

ST_Length

この関数は、複数折れ線を受け取り、倍精度数としてすべての折れ線要素の長さの累計を戻します。詳細については、240ページの『ST_Length』を参照してください。

ST_IsClosed

この関数は、複数折れ線を受け取り、その複数折れ線が閉じている場合は 1 (TRUE)、それ以外の場合は 0 (FALSE) を戻します。詳細については、231ページの『ST_IsClosed』を参照してください。

ST_NumGeometries

同種集合を受け取り、その中に含まれている基本図形要素の数を戻します。詳細については、250ページの『ST_NumGeometries』を参照してください。

ST_GeometryN

同種集合および指数を受け取り、何番目か (任意) の基本図形を戻します。詳細については、221ページの『ST_GeometryN』を参照してください。

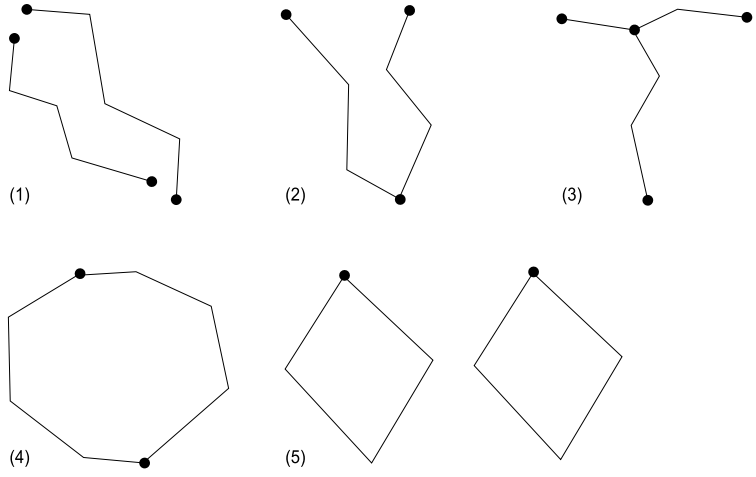
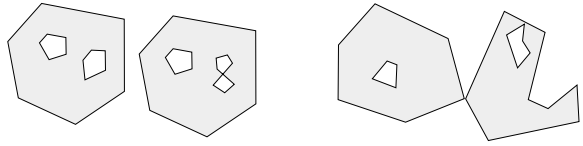


図 12. 複数折れ線.

1. 2 つの折れ線要素にある 4 つの端点によって境界が定義されている、単純な複数折れ線。
2. 単純な複数折れ線 (折れ線要素の端点だけで交わっているため)。境界は、2 つの交わっていない端点によって定義されています。
3. 単純でない複数折れ線 (折れ線のうちの 1 つの内部で交わっているため)。この複数折れ線の境界は、交わっているポイントを含む、4 つの端点で定義されています。
4. 単純な閉じていない複数折れ線。要素である折れ線が閉じていないので、この複数折れ線は閉じていません。また、要素である折れ線の中に内部で交差しているものがないため、この複数折れ線は単純です。
5. 単純な閉じている複数折れ線。すべての要素が閉じているので、この複数折れ線は閉じています。また、内部で交差している要素がないので、この複数折れ線は単純です。

複数ポリゴン

複数ポリゴンの境界の長さは、その要素の外部および内部のリングの累計になります。複数ポリゴンの内部は、要素であるポリゴンの累積した内部として定義されます。複数ポリゴンの要素の境界は、接点でのみ交わることができます。スーパークラス図形から継承される他の特性に加えて、複数ポリゴンには面積があります。複数ポリゴンは、森林地域や、一続きの島々などの連続している土地の区画などの地形を定義します。



(1) (2)

図 13. 複数ポリゴン.

1. この複数ポリゴンには、ポリゴンの要素が 2 つあります。境界は、外部リング 2 つと内部リング 3 つによって定義されています。
2. この複数ポリゴンには、ポリゴンの要素が 2 つあります。境界は、外部リング 2 つと内部リング 2 つによって定義されています。 2 つのポリゴン要素が接点で交わっています。

複数ポリゴンを操作する関数を以下に示します。

ST_Area

この関数は、複数ポリゴンを受け取り、倍精度数として多角形の要素の累計の面積を戻します。詳細については、189ページの『ST_Area』を参照してください。

ST_Centroid

この関数は、複数ポリゴンを受け取り、その地理的な中心にあるポイントを戻します。詳細については、197ページの『ST_Centroid』を参照してください。

ST_NumGeometries

同種集合を受け取り、その中に含まれている基本図形要素の数を戻します。詳細については、250ページの『ST_NumGeometries』を参照してください。

ST_GeometryN

同種集合および指数を受け取り、何番目か (任意) の基本図形を戻します。詳細については、221ページの『ST_GeometryN』を参照してください。

関係と比較を示し、図形を生成し、値の形式を変換する関数

ここまでの節では、地理情報関数に関する以下の区分を紹介しました。

- 図形の特性に関連した関数
- 特定の図形に関連した関数

この節では、さらに次の 3 つの区分を紹介します。

- 地形の関連または比較の仕方を判別する関数

- 新しい図形を生成する関数
- 図形の値を、インポートまたはエクスポートされた形式に変換する関数

地形間の関係および比較を示す関数

いくつかの地理情報関数は、地形が相互に関連または比較する仕方についての情報を戻します。その大半は*述部* と呼ばれる関数で、ブール関数です。ここでは、一般的な述部関数について記した後、各関数を個別に説明します。

述部関数

述部関数は、比較が関数の基準に適合すれば 1 (TRUE)、比較が失敗すれば 0 (FALSE) を戻します。地理関係をテストする述部は、2 つの図形を比較します。この 2 つの図形は、タイプや次元が異なっていても構いません。

述部は、送られた図形の X および Y 座標を比較します。 Z 座標および測定値 (ある場合) は無視されます。これにより、Z 座標または測定値がある図形と、それらがない図形を比較することができます。

Dimensionally Extended 9 Intersection Model (DE-9IM)[1] は、タイプおよび次元の異なる図形同士の地理関係 (のようなもの) を定義する、数学的な手法です。このモデルでは、結果の交差部分の次元を考慮に入れた上で、図形のすべてのタイプの間の地理関係を、内部、境界、および外部の交差部分として表現しています。

a と *b* という図形があるとします。 I(*a*)、B(*a*)、および E(*a*) は、それぞれ *a* の内部、境界、外部を表します。そして I(b)、B(b)、および E(b) は、 *b* の内部、境界、および外部を表します。 I(a)、B(a)、および E(a) と、 I(b)、B(b)、および E(b) との交差部分は、3 x 3 の行列になります。それぞれの交差部分は、異なる次元の図形になる場合があります。たとえば、2 つのポリゴンの境界の交差部分は、ポイントおよび折れ線で構成されています。この場合、dim 関数は最大次元として 1 を戻します。

---

1. DE-91M は、Clementini と Felice によって開発されたもので、 Egenhofer と Herring による 9 Intersection Model を次元の点で拡張しています。 DE-91M は、4 人の開発者 Clementini、Eliseo、Di Felice、van Osstrom が共同開発したものです。彼らはそのモデルを *Advances in Spatial Database--Third International Symposium. SSD '93*. LNCS 692. Pp. 277-295 の 『A Small Set of Formal Topological Relationships Suitable for End-User Interaction』 D. Abel および B.C. Ooi (編) で発表しました。 Springer-Verlag Singapore (1993) Egenhofer M.J. および Herring, J. による 9 Intersection モデルは、 *Tech. Report, Department of Surveying Engineering*, University of Maine, Orono, ME 1991 の 『Categorizing binary topological relationships between regions, lines, and points in geographic databases』で発表されました。

dim 関数は 1、0、1 または 2 の値を戻します。 1 は ヌル集合または dim(null) (交差部分が見つからない場合に戻される) に対応しています。

	内部	境界	外部
内部	dim(I(a) ∩ I(b))	dim(I(a) ∩ B(b))	dim(I(a) ∩ E(b))
境界	dim(B(a) ∩ I(b))	dim(B(a) ∩ B(b))	dim(B(a) ∩ E(b))
外部	dim(E(a) ∩ I(b))	dim(E(a) ∩ B(b))	dim(E(a) ∩ E(b))

地理関係の述部の結果は、述部の結果を DE-9IM の許容値を表すパターン行列と比較することによって、理解または検査することができます。

パターン行列には、交差部分の行列のセルごとの許容値が含まれています。可能なパターン値は、以下のとおりです。

T 交差がなければなりません (dim = 0、1、または 2)。

F 交差があってはなりません (dim = -1)。

***** 交差があるかどうかは関係ありません (dim = -1、0、1、または 2)。

0 交差がなければならず、最大次元は 0 でなければなりません (dim = 0)。

1 交差がなければならず、最大次元は 1 でなければなりません (dim = 1)。

2 交差がなければならず、最大次元は 2 でなければなりません (dim = 2)。

たとえば、ST_Within 述部の場合、以下のパターン行列に値 T、F、* が含まれます。

表 40. ST_Within の行列. 図形の組み合わせの ST_Within のパターン行列

		b		
		内部	**境界**	**外部**
a	**内部**	T	*	F
	境界	*	*	F
	外部	*	*	*

両方の図形の内部が交差する場合、および *a* の内部および境界が *b* の外部と交差しない場合、 ST_Within 述部は TRUE を戻します。ほかのすべての条件では、関係ありません。

各述部には最低 1 つのパターン行列がありますが、一部の述部では、さまざまな図形タイプの組み合わせの関係を記述するために、複数のパターン行列が必要です。

ST_Equals

ST_Equals は、同じタイプの 2 つの図形が同一の X、Y 座標値を持っている場合、 1 (TRUE) を戻します。

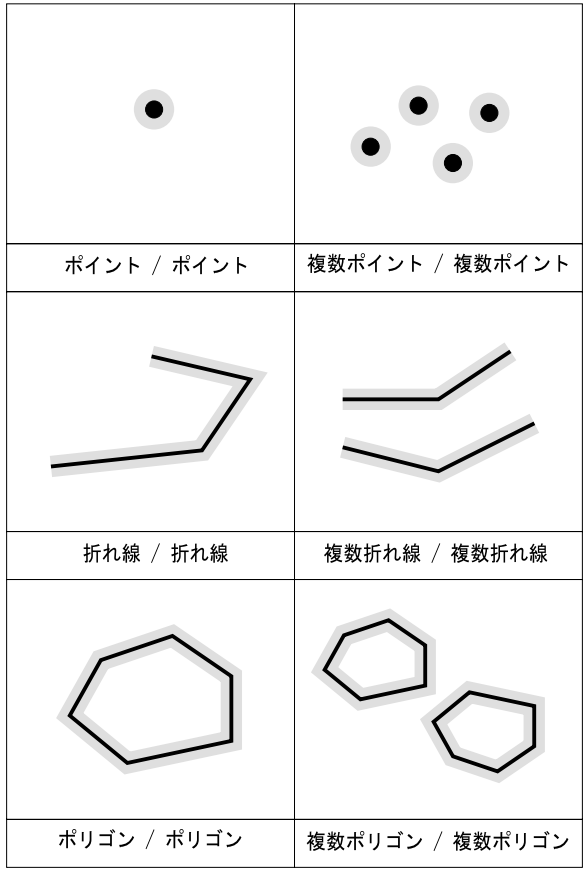


図 14. ST_Equals. X、Y 座標が一致している場合、図形は同一です。

表 41. equals の行列. equals の DE-9IM パターン行列では、内部が交差することと、どちらの図形の内部または境界も他方の外部と交差しないことが保証されています。

		b		
		内部	境界	外部
a	内部	T	*	F
	境界	*	*	F
	外部	F	F	*

詳細については、215ページの『ST_Equals』を参照してください。

ST_OrderingEquals

ST_OrderingEquals は 2 つの図形を比較し、図形が等しく、座標が同じ順序であれば 1 (TRUE) を戻し、そうでなければ 0 (FALSE) を戻します。詳細については、253ページの『ST_OrderingEquals』を参照してください。

ST_Disjoint

ST_Disjoint は、2 つの図形の交差部分が空のセットの場合、1 (TRUE) を戻します。

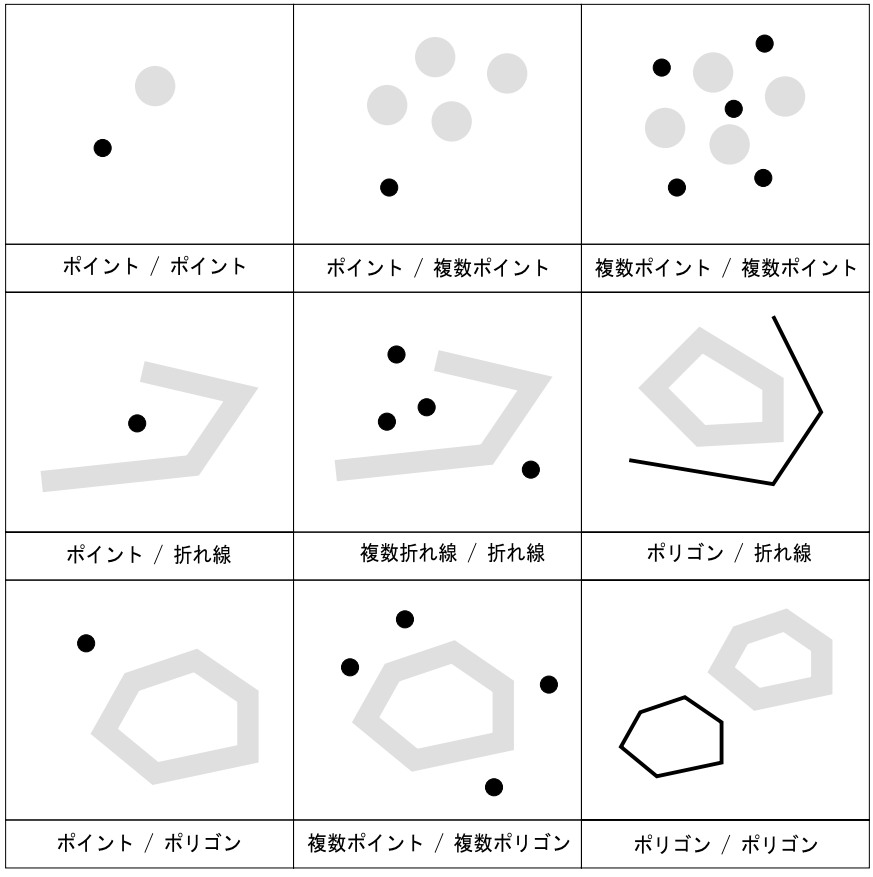


図 15. ST_Disjoint. いかなる形でも互いに交わっていない場合、図形は結合していません。

表 42. ST_Disjoint の行列. ST_Disjoint 述部のパターン行列は、単にどちらかの図形の内部も境界も交差しないことを示しています。

		b		
		内部	**境界**	**外部**
a	**内部**	F	F	*
	境界	F	F	*
	外部	*	*	*

詳細については、209ページの『ST_Disjoint』を参照してください。

ST_Intersects

ST_Intersects は、交差部分が空のセットにならない場合に 1 (TRUE) を戻します。 ST_Intersects は、ST_Disjoint とまったく逆の結果を戻します。

以下のパターン行列のいずれかの条件が TRUE を戻す場合、 ST_Intersects 述部は TRUE を戻します。

表 43. ST_Intersects の行列 (1). 両方の図形の内部が交わっている場合、ST_Intersects 述部は TRUE を戻します。

		b		
		内部	**境界**	**外部**
a	**内部**	T	*	*
	境界	*	*	*
	外部	*	*	*

表 44. ST_Intersects の行列 (2). 最初の図形の境界と 2 番目の図形の境界が交わっている場合、 ST_Intersects 述部は TRUE を戻します。

		b		
		内部	**境界**	**外部**
a	**内部**	*	T	*
	境界	*	*	*
	外部	*	*	*

表 45. ST_Intersects の行列 (3). 最初の図形の境界と 2 番目の図形の内部が交わっている場合、 ST_Intersects 述部は TRUE を戻します。

		b		
		内部	**境界**	**外部**
a	**内部**	*	*	*
	境界	T	*	*
	外部	*	*	*

表 46. ST_Intersects の行列 (4). どちらかの図形の境界が交わっている場合、ST_Intersects 述部は TRUE を戻します。

		b		
		内部	**境界**	**外部**
a	**内部**	*	*	*
	境界	*	T	*
	外部	*	*	*

詳細については、230ページの『ST_Intersects』を参照してください。

EnvelopesIntersect

この関数は、2 つの図形のエンベロープが交差する場合に 1 (TRUE) を戻します。これは、ST_Intersects (`ST_Envelope(g1),ST_Envelope(g2)`) を効率的に実装する便利な関数です。詳細については、172ページの『EnvelopesIntersect』を参照してください。

ST_Touches

ST_Touches は、2 つの図形に共通するポイントのうち、両方の図形の内部で交差するものが存在しない場合に、1 (TRUE) を戻します。少なくとも 1 つの図形が、折れ線、ポリゴン、複数折れ線、または複数ポリゴンのいずれかでなければなりません。

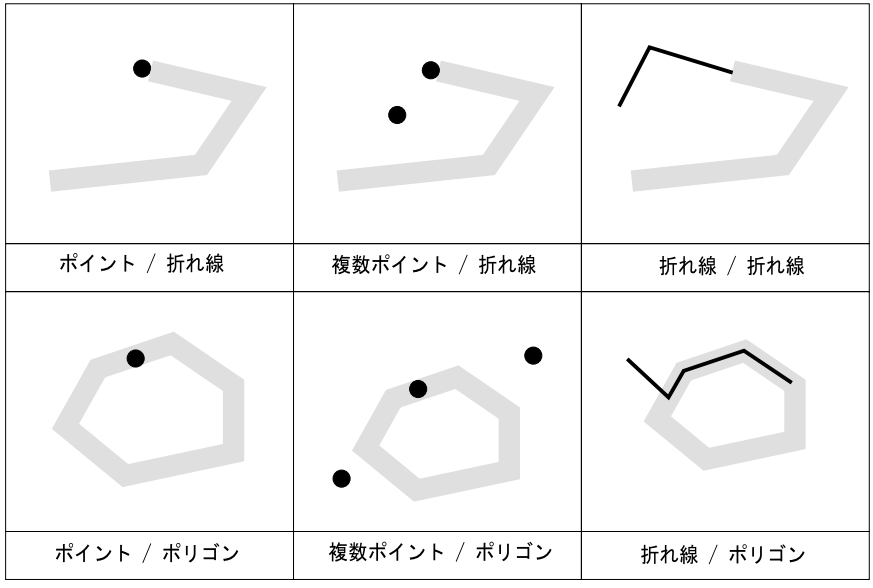


図 16. ST_Touches

以下のパターン行列は、図形の内部が交わらず、どちらかの図形の境界が他方の内部または境界と交わる場合に、 ST_Touches 述部が TRUE を戻すことを示しています。

表 47. ST_Touches の行列 (1)

		b		
		内部	**境界**	**外部**
a	**内部**	F	T	*
	境界	*	*	*
	外部	*	*	*

表 48. ST_Touches の行列 (2)

		b		
		内部	**境界**	**外部**
a	**内部**	F	*	*
	境界	T	*	*
	外部	*	*	*

表 49. ST_Touches の行列 (3)

		b		
		内部	境界	外部
a	内部	F	*	*
	境界	*	T	*
	外部	*	*	*

詳細については、272ページの『ST_Touches』を参照してください。

ST_Overlaps

ST_Overlaps は、同じ次元の 2 つの図形を比較します。この関数は、交差部分のセットがどちらとも違う図形であるものの、同じ次元である場合に、1 (TRUE) を戻します。

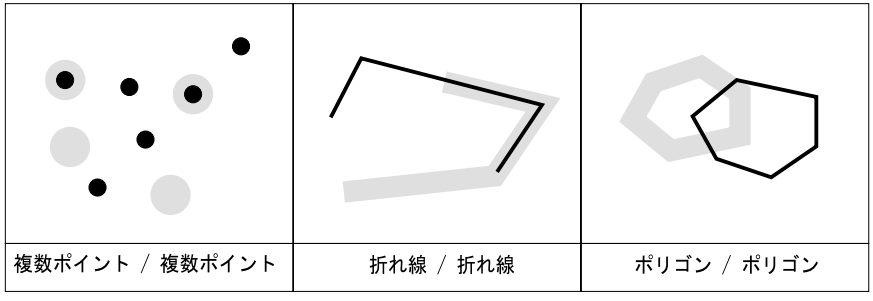


図 17. ST_Overlaps

表50 のパターン行列は、重なっているポリゴン / ポリゴン、複数ポイント / 複数ポイント、および複数ポリゴン / 複数ポリゴンの組み合わせに適用されます。これらの組み合わせで、両方の図形の内部が他方の内部および外部と交差する場合に、オーバーレー述部は TRUE を戻します。

表 50. ST_Overlaps の行列 (1)

		b		
		内部	境界	外部
a	内部	T	*	T
	境界	*	*	*
	外部	T	*	*

155ページの表51 のパターン行列は、折れ線 / 折れ線および複数折れ線 / 複数折れ線の組み合わせに適用されます。この場合、図形の交差部分は次元が 1 である図形 (別の折れ線) でなければなりません。内部の交差部分の次元が 1 で

あれば、ST_Overlaps 述部は FALSE を戻しますが、 ST_Crosses 述部は TRUE を戻します。

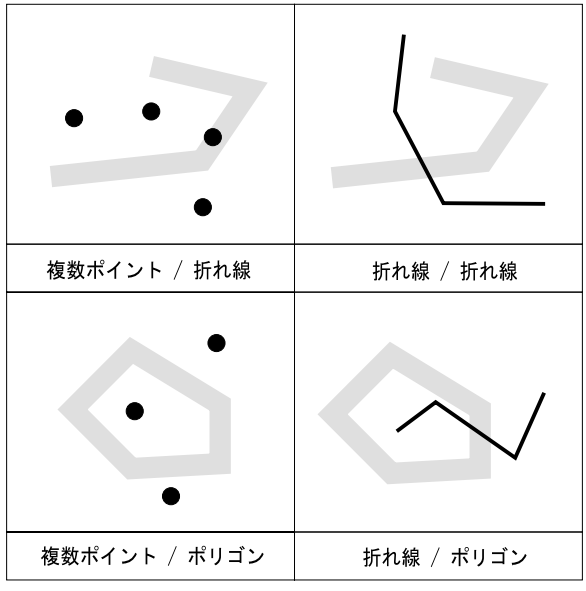
表 51. ST_Overlaps の行列 (2)

		b		
		内部	境界	外部
a	内部	1	*	T
	境界	*	*	*
	外部	T	*	*

詳細については、254ページの『ST_Overlaps』を参照してください。

ST_Crosses

ST_Crosses は、交差部分が 2 つのソース図形の最大次元よりも 1 つ小さい次元の図形になり、交差部分のセットが両方のソース図形の中にある場合に、1 (TRUE) を戻します。 ST_Crosses が 1 (TRUE) を戻すのは、複数ポイント / ポリゴン、複数ポイント / 折れ線、折れ線 / 折れ線、折れ線 / ポリゴン、および折れ線 / 複数ポリゴンの組み合わせを比較する場合だけです。



156ページの表52 のパターン行列は、複数ポイント / 折れ線、複数ポイント / 複数折れ線、複数ポイント / ポリゴン、複数ポイント / 複数ポリゴン、折れ線 / ポリゴン、および折れ線 / 複数ポリゴンの組み合わせに適用されます。この行列は、内部が交差しなければならないことと、 1 番目の図形 (図形 *a*) の内部が、 2 番目の図形 (図形 *b*) の外部と交差しなければならないことを示して

います。

表 52. ST_Crosses の行列 (1)

		b		
		内部	境界	外部
a	内部	T	*	T
	境界	*	*	*
	外部	*	*	*

表53 のパターン行列は、折れ線 / 折れ線、折れ線 / 複数折れ線、および複数折れ線 / 複数折れ線の組み合わせに適用されます。この行列は、内部の交差部分の次元が 0 (ポイントで交差する) でなければならないことを示しています。この交差部分の次元が 1 (折れ線で交差する) であれば、 ST_Crosses 述部は FALSE を戻しますが、ST_Overlaps 述部は TRUE を戻します。

表 53. ST_Crosses の行列 (2)

		b		
		内部	境界	外部
a	内部	0	*	*
	境界	*	*	*
	外部	*	*	*

詳細については、204ページの『ST_Crosses』を参照してください。

ST_Within

ST_Within は、最初の図形が完全に 2 番目の図形の中にある場合に、1 (TRUE) を戻します。 ST_Within は、ST_Contains とまったく逆の結果を戻します。

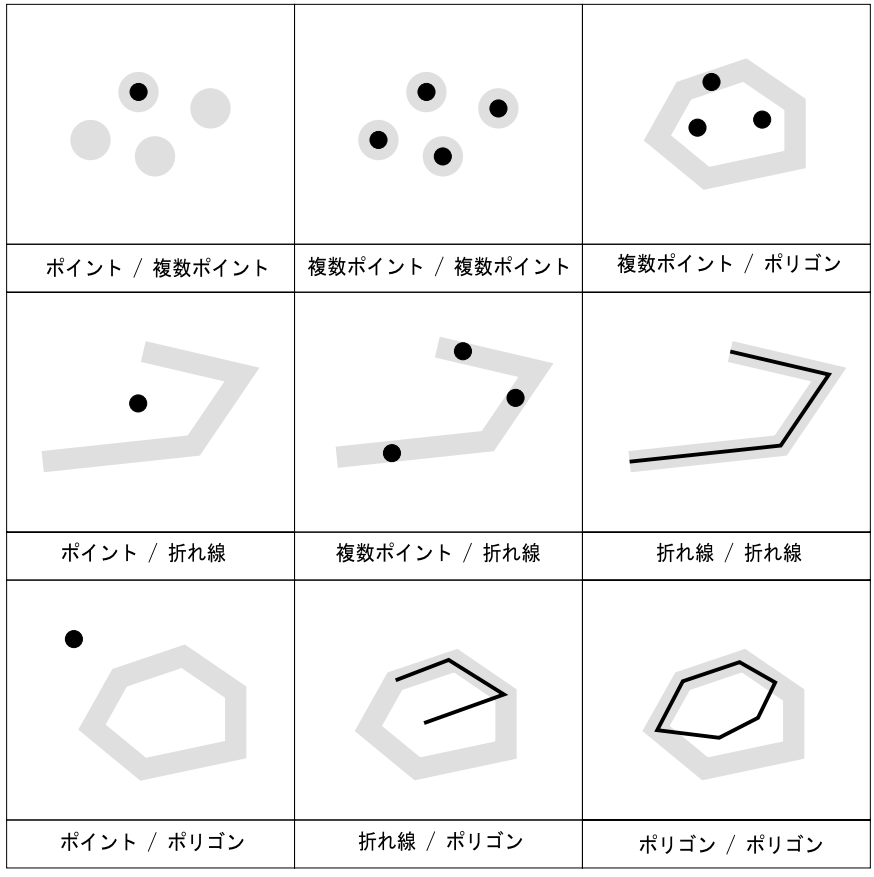


図 18. ST_Within

この ST_Within 述部のパターン行列は、両方の図形の内部が交差しなければならないことと、 1 番目の図形 (図形 *a*) の内部と境界が、 2 番目の図形 (図形 *b*) の外部と交差してはならないことを示しています。

表 54. ST_Within の行列

		b		
		内部	**境界**	**外部**
a	**内部**	T	*	F
	境界	*	*	F
	外部	*	*	*

詳細については、275ページの『ST_Within』を参照してください。

ST_Contains

ST_Contains は、2 番目の図形が完全に最初の図形の中にある場合に、1 (TRUE) を戻します。 ST_Contains 述部は、ST_Within 述部とまったく逆の結果を戻します。

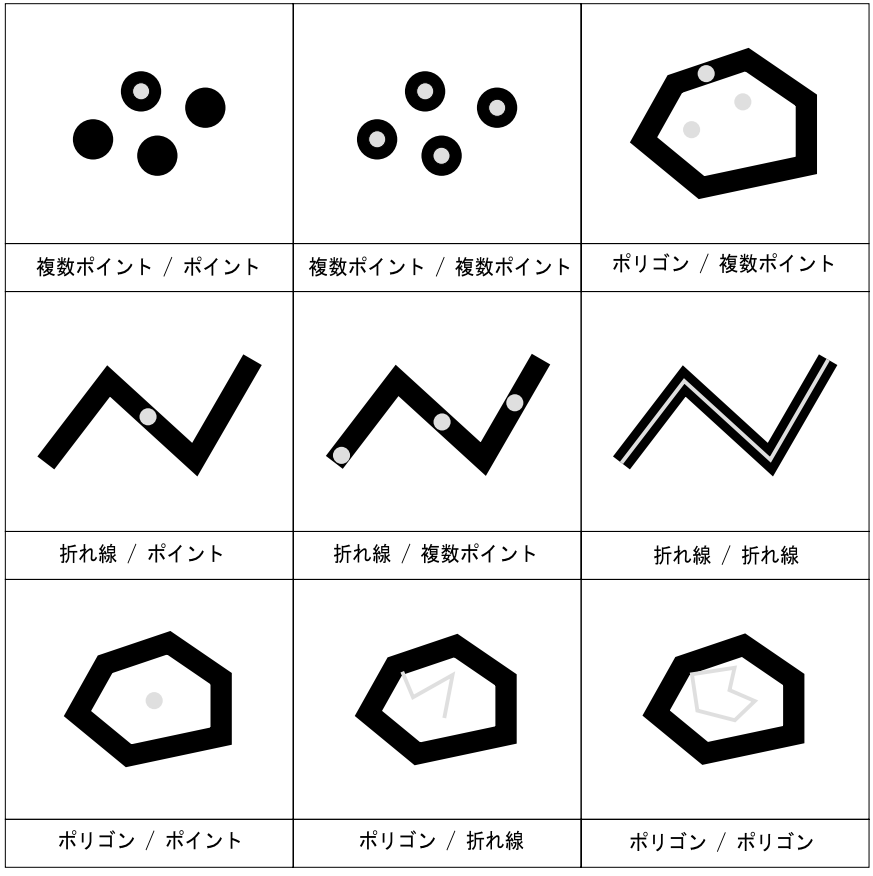


図 19. ST_Contains

ST_Contains 述部のパターン行列は、両方の図形の内部が交差しなければならないことと、 2 番目の図形 (図形 *b*) の内部と境界が、 1 番目の図形 (図形 *a*) の外部と交差してはならないことを示しています。

表 55. ST_Contains の行列

		b		
		内部	境界	外部
a	内部	T	*	*
	境界	*	*	*
	外部	F	F	*

詳細については、198ページの『ST_Contains』を参照してください。

ST_Relate

ST_Relate 関数は 2 つの図形を比較し、図形が DE-91M パターン行列ストリングで指定された条件に適合する場合は 1 (TRUE) を戻し、そうでなければ 0 (FALSE) を戻します。詳細については、266ページの『ST_Relate』を参照してください。

ST_Distance

ST_Distance 関数は、2 つの分離している地形の隔たりの最短距離を報告します。地形が分離していない場合、この関数は最短距離として 0 を報告します。

たとえば、ST_Distance は、飛行機が 2 つの地点間を飛ぶのに必要な最短距離を報告します。この情報を示しているのが図20 です。

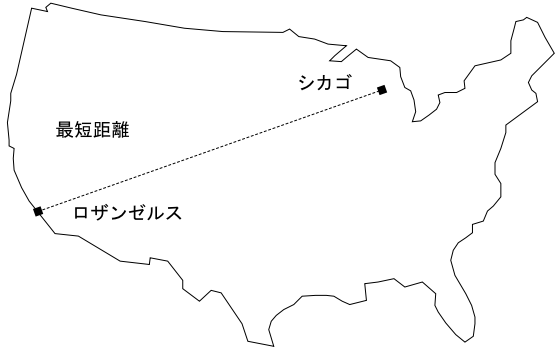


図 20. 2 都市間の最短距離. ST_Distance は、ロサンゼルスとシカゴの場所を示す座標を入力データとして受け取り、 2 地点間の最短距離を示す値を戻します。

詳細については、211ページの『ST_Distance』を参照してください。

既存の図形から新しい図形を生成する関数

DB2 地理情報エクステンダーは、既存の図形から新しい図形を生成する述部関数と変換関数を提供します。

ST_Intersection

ST_Intersection 関数は、2 つの図形の交差部分のセットを戻します。常に交差部分セットは、次元がソース図形の最小の次元と同じである集合として戻されます。たとえば、ポリゴンと交差する折れ線の場合、ST_Intersection 関数は、折れ線のうちポリゴンの内部および境界と共通な部分で構成される複数折れ線を戻します。ソース折れ線が複数の連続していない区画でポリゴンと交差している場合、この複数折れ線には複数の折れ線が含まれます。図形が交差していない、または交差部分の次元が両方のソース図形の次元よりも低い場合、空の図形が戻されます。

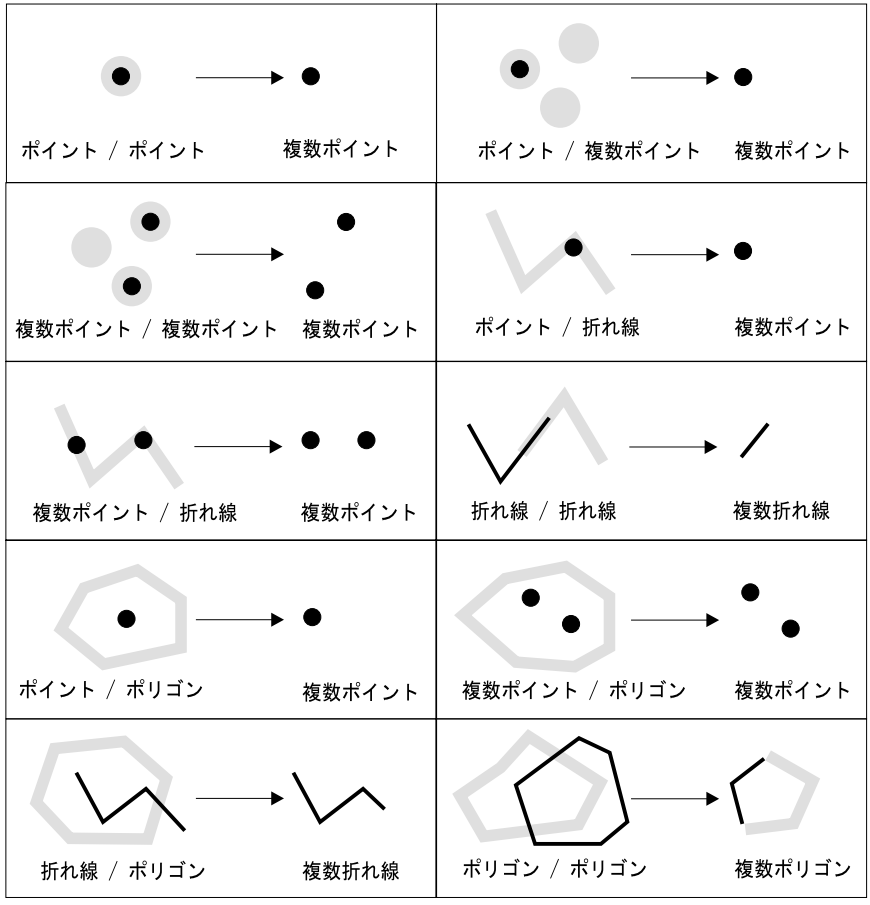


図 21. ST_Intersection. ST_Intersection 関数の例。

詳細については、228ページの『ST_Intersection』を参照してください。

ST_Difference

ST_Difference 関数は、最初の図形のうち、2 番目の図形と交差しない部分を戻します。これはスペースの論理 AND NOT です。 ST_Difference 関数は同じ次元の図形だけを操作し、ソース図形と同じ次元を持つ集合を戻します。ソース図形が同じである場合、空の図形が戻されます。

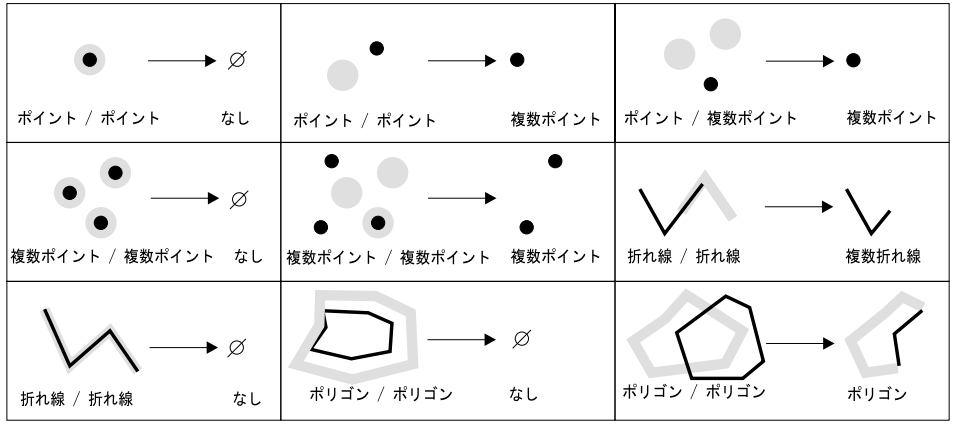


図 22. ST_Difference

詳細については、206ページの『ST_Difference』を参照してください。

ST_Union

ST_Union 関数は、2 つの図形の和集合のセットを戻します。これは、スペースの論理 OR です。ソース図形の次元は同じでなければなりません。
ST_Union は結果を必ず集合として戻します。

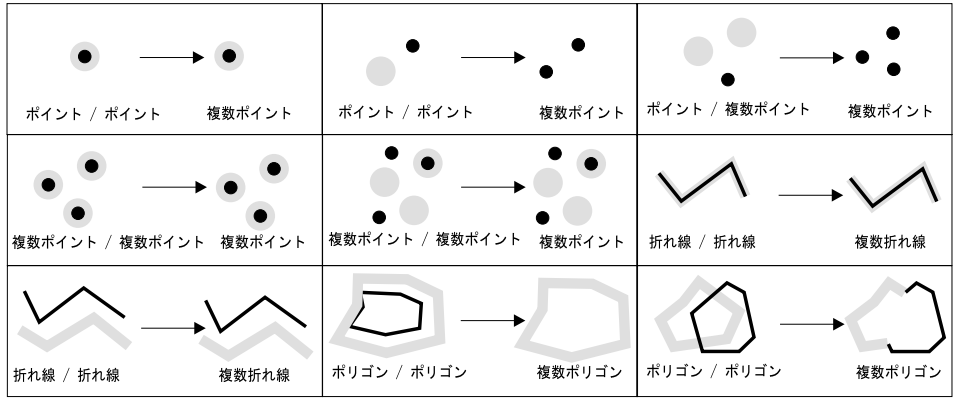


図 23. ST_Union

詳細については、274ページの『ST_Union』を参照してください。

ST_Buffer

ST_Buffer 関数は、指定の距離を置いて囲んだ図形を生成します。最初の図形に緩衝地帯 (buffer) が付けられている場合や、緩衝地帯のポリゴンがすべて重なるほど集合の要素が近くにある場合には、ポリゴンが戻されます。ただし、緩衝地帯が付いている集合で、緩衝地帯のポリゴンの要素の間に十分の距離がある場合、 ST_Buffer 関数は複数ポリゴンを戻します。

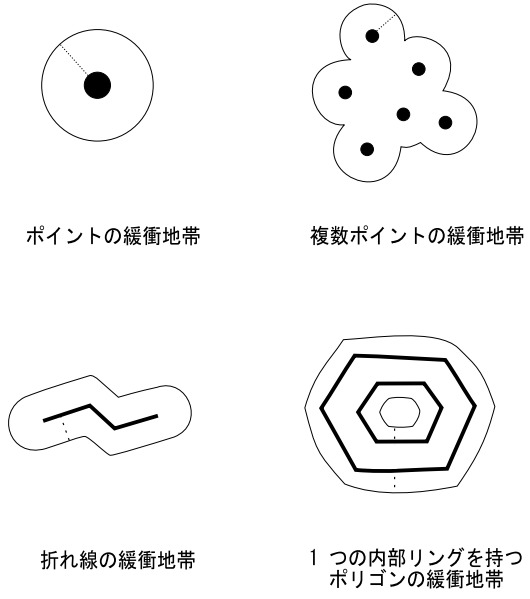


図 24. ST_Buffer

ST_Buffer 関数は、正と負の両方の距離を受け入れますが、負の緩衝地帯が適用されるのは、2 次元の図形 (ポリゴンおよび複数ポリゴン) だけです。ソース図形の次元が 2 よりも低い場合 (ポリゴンまたは複数ポリゴン以外のすべての図形) はいつでも、緩衝地帯の距離の絶対値が使用されます。

一般に、外部リングの場合、緩衝地帯の距離が正の場合はソース図形の中心から離れる方向にポリゴンのリングが生成され、距離が負の緩衝地帯の場合はポリゴンまたは複数ポリゴンが中心の方向に生成されます。ポリゴンまたは複数ポリゴンの内部リングの場合、緩衝地帯の距離が正なら緩衝地帯のリングが中心に向かって生成され、緩衝地帯の距離が負なら緩衝地帯のリングが中央とは逆の方向に生成されます。

緩衝地帯を付ける処理では、重なり合うポリゴンが結合されます。負の距離が、ポリゴンの内部の最長部分の半分より大きい場合、空の図形が生成されます。

詳細については、195ページの『ST_Buffer』を参照してください。

LocateAlong

測定値を持つ図形の場合、特定の測定値の場所を LocateAlong 関数を使用して見つけることができます。 LocateAlong は、複数ポイントとして位置を戻します。ソース図形の次元が 0 (すなわち、ポイントまたは複数ポイント) の場合、それらは完全に一致していなければならず、測定値が一致しているそれらのポイントが複数ポイントとして戻されます。ただし、ソース図形の次元が 0 より大きい場合は、位置が補間されます。たとえば、入力した測定値が 5.5 で、折れ線の頂点の測定値がそれぞれ 3、4、5、6、および 7 である場合、測定値が 5 および 6 の頂点のちょうど中間に位置する補間されたポイントが戻されます。

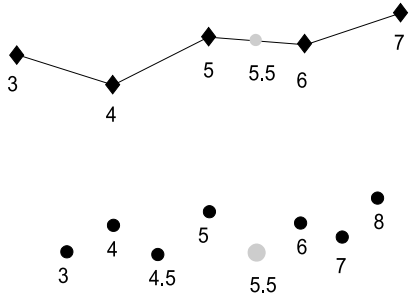


図 25. LocateAlong

詳細については、177ページの『LocateAlong』を参照してください。

LocateBetween

LocateBetween 関数は、測定値を持つソース図形の 2 つの測定値の間にある一連のパスまたは位置を戻します。ソース図形の次元が 0 の場合、LocateBetween は、2 つのソース測定値の間に測定値があるすべてのポイントを含む複数ポイントを戻します。次元が 0 よりも大きいソース図形について、LocateBetween は、パスを補間できれば複数折れ線を戻します。それができなければ、LocateBetween はポイントの位置を含む複数ポイントを戻します。LocateBetween がパスを補間できない場合や、測定値間の位置を見つけられない場合は、常に空のポイントが戻されます。 LocateBetween は図形の包括的な

検索を行います。そのため、図形の測定値は必ず起点の測定値*以上* で、なおかつ終点の測定値*以下* でなければなりません。

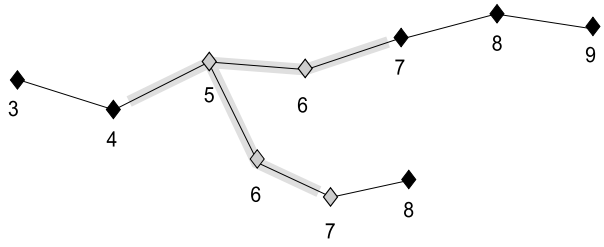


図 26. LocateBetween

詳細については、179ページの『LocateBetween』を参照してください。

ST_ConvexHull

ST_ConvexHull 関数は、最低でも 3 つの頂点が凸を形成している任意の図形の凸包 (convex hull) ポリゴンを戻します。図形の頂点が凸を形成していない場合、ST_ConvexHull はヌルを戻します。 ST_ConvexHull は、ポイントの集合から TIN ネットワークを作成する際に行うモザイク化の最初のステップでしばしば使用されます。

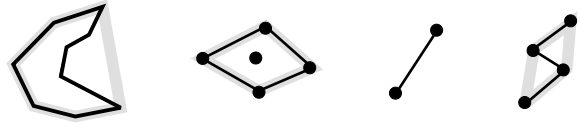


図 27. ST_ConvexHull

詳細については、200ページの『ST_ConvexHull』を参照してください。

ST_Polygon

折れ線からポリゴンを生成します。詳細については、265ページの『ST_Polygon』を参照してください。

図形の値の形式を変換する関数

DB2 地理情報エクステンダーは 3 つの GIS データ交換形式をサポートしています。

- 事前割り当てテキスト表現
- 事前割り当てバイナリー表現
- ESRI バイナリー形状表現

事前割り当てテキスト表現

DB2 地理情報エクステンダーには、テキスト記述から図形を生成する関数がいくつかあります。

ST_WKTToSQL
任意の図形タイプのテキスト表現から図形を生成します。地理情報参照システムの識別子は指定しないでください。 詳細については、279ページの『ST_WKTToSQL』を参照してください。

ST_GeomFromText
任意の図形タイプのテキスト表現から図形を生成します。地理情報参照システムの識別子を指定する必要があります。 詳細については、218ページの『ST_GeometryFromText』を参照してください。

ST_PointFromText
ポイントのテキスト表現からポイントを作成します。詳細については、257ページの『ST_PointFromText』を参照してください。

ST_LineFromText
折れ線のテキスト表現から折れ線を作成します。詳細については、242ページの『ST_LineFromText』を参照してください。

ST_PolyFromText
ポリゴンのテキスト表現からポリゴンを作成します。詳細については、262ページの『ST_PolyFromText』を参照してください。

ST_MPointFromText
複数ポイントの表現から複数ポイントを作成します。詳細については、246ページの『ST_MPointFromText』を参照してください。

ST_MLineFromText
複数折れ線の表現から複数折れ線を作成します。詳細については、244ページの『ST_MLineFromText』を参照してください。

ST_MPolyFromText
複数ポリゴンの表現から複数ポリゴンを作成します。詳細については、248ページの『ST_MPolyFromText』を参照してください。

テキスト表現は ASCII ストリングです。このため、図形を ASCII テキスト形式で交換することができます。これらの関数では、バイナリー表現をマップするための特殊なプログラム構造を定義する必要はありません。そのため、3GL または 4GL プログラムのどちらでも使用することができます。

ST_AsText 関数は、既存の図形値をテキスト表現に変換します。詳細については、192ページの『ST_AsText』を参照してください。

事前割り当てテキスト表現の詳細については、293ページの『OGC による事前割り当てテキスト表現』を参照してください。

事前割り当てバイナリー表現

DB2 地理情報エクステンダーには、事前割り当てバイナリー (WKB) 表現から図形を生成する関数がいくつかあります。

ST_WKBToSQL

任意の図形タイプの事前割り当てバイナリー表現から図形を作成します。 地理情報参照システムの識別子は指定しないでください。 詳細については、277ページの『ST_WKBToSQL』を参照してください。

ST_GeomFromWKB

任意の図形タイプの事前割り当てバイナリー表現から図形を作成します。 地理情報参照システムの識別子を指定する必要があります。 詳細については、219ページの『ST_GeomFromWKB』を参照してください。

ST_PointFromWKB

ポイントの事前割り当てバイナリー表現からポイントを作成します。詳細については、258ページの『ST_PointFromWKB』を参照してください。

ST_LineFromWKB

折れ線の事前割り当てバイナリー表現から折れ線を作成します。詳細については、243ページの『ST_LineFromWKB』を参照してください。

ST_PolyFromWKB

ポリゴンの事前割り当てバイナリー表現からポリゴンを作成します。詳細については、263ページの『ST_PolyFromWKB』を参照してください。

ST_MPointFromWKB

複数ポイントの事前割り当てバイナリー表現から複数ポイントを作成します。詳細については、247ページの『ST_MPointFromWKB』を参照してください。

ST_MLineFromWKB

複数折れ線の事前割り当てバイナリー表現から複数折れ線を作成します。詳細については、245ページの『ST_MLineFromWKB』を参照してください。

ST_MPolyFromWKB

複数ポリゴンの事前割り当てバイナリー表現から複数ポリゴンを作成します。詳細については、249ページの『ST_MPolyFromWKB』を参照してください。

事前割り当てバイナリー表現は、連続する一連のバイトです。これにより、ODBC クライアントと SQL データベースの間で図形をバイナリー形式で交換することができます。これらの関数では、バイナリー表現をマップする C 構造を定義する必要があります。そのため、3GL プログラムで使用するよう意図されており、4GL 環境での使用には適していません。

ST_AsBinary 関数は、既存の図形値を事前割り当てバイナリー表現に変換します。詳細については、191ページの『ST_AsBinary』を参照してください。

事前割り当てバイナリー表現の詳細については、298ページの『OGC による事前割り当てバイナリー (WKB) 表現』を参照してください。

ESRI 形状表現

DB2 地理情報エクステンダーには、ESRI 形状表現から図形を生成する関数がいくつかあります。 ESRI 形状表現は、テキストおよび事前割り当てバイナリー表現でサポートされる 2 次元表現に加えて、 Z 座標と測定値をサポートします。

ShapeToSQL

任意の図形タイプの形状表現から図形を作成します。地理情報参照システムの識別子は指定しないでください。 詳細については、188ページの『ShapeToSQL』を参照してください。

GeometryFromShape

任意の図形タイプの形状表現から図形を作成します。地理情報参照システムの識別子を指定する必要があります。詳細については、171ページの『GeometryFromShape』を参照してください。

PointFromShape

ポイント形状からポイントを作成します。詳細については、185ページの『PointFromShape』を参照してください。

LineFromShape

折れ線形状から折れ線を作成します。詳細については、175ページの『LineFromShape』を参照してください。

PolyFromShape

折れ線形状からポリゴンを作成します。詳細については、186ページの『PolyFromShape』を参照してください。

MPointFromShape

複数ポイント形状から複数ポイントを作成します。詳細については、183ページの『MPointFromShape』を参照してください。

MLineFromShape

複数パートの折れ線形状から複数折れ線を作成します。詳細については、182ページの『MLine FromShape』を参照してください。

MPolyFromShape

複数ポリゴン形状から複数ポリゴンを作成します。詳細については、184ページの『MPolyFromShape』を参照してください。

これらの関数の一般的な構文は同じです。最初の引き数は、BLOB データ・タイプとして入力される形状表現です。 2 番目の引き数は、図形に割り当てられる地理情報参照 ID です。たとえば、GeometryFromShape 関数は以下のような構文になります。

```
GeometryFromShape(shapegeometry, SRID)
```

バイナリー表現をマップできるよう、これらの形状関数では C 構造を定義する必要があります。そのため、3GL プログラムで使用するよう意図されており、4GL 環境での使用には適していません。

AsBinaryShape 関数は、図形値を ESRI 形状表現に変換します。詳細については、170ページの『AsBinaryShape』を参照してください。

形状表現の詳細については、302ページの『ESRI 形状表現』を参照してください。

第14章 SQL 照会のための地理情報関数

この章には、地理情報データを照会するときに呼び出すことのできる関数がリストされています。それぞれの関数が 1 つのセクションで取り上げられ、構文、戻りタイプ、およびコード例が示されます。この章の例では、 1 つまたは複数の列が地理情報列として定義される CREATE TABLE ステートメントが含められる場合があります。

以下に示す考慮事項が地理情報関数に適用されます。

- この章の例は、ライブラリー名 db2gse で修飾されます。各地理情報関数とタイプを db2gse で明示的に修飾する代わりに、関数パスが db2gse を含むように設定することができます。
- 地理情報列にデータを挿入する前に、以下を行ってください。
  - udf_mem_sz パラメーターを増やす必要があります。推奨されている初期設定は 2048 です。 2048 で十分でない場合は、 udf_mem_sz パラメーターを 256 ずつ増やしてください。
  - 地理情報列をレイヤーとして登録する必要があります。地理情報列をレイヤーとして登録する方法については、 35ページの『第4章 地理情報列を定義し、レイヤーとして登録して、ジオコーダーを使用して保守できるようにする』を参照してください。

AsBinaryShape

AsBinaryShape は図形オブジェクトを引き数とし、BLOB を戻します。

構文

db2gse.AsBinaryShape(g db2gse.ST_Geometry)

戻りタイプ

BLOB(1m)

例

下記のコードの一部は、 AsBinaryShape 関数が SENSITIVE_AREAS 表のゾーン・ポリゴンを形状ポリゴンに変換する様子を示しています。これらの形状ポリゴンは、アプリケーションの draw_polygon 関数に渡されて表示されます。

```
/* Create the SQL expression. */
strcpy(sqlstmt, "select db2gse.AsBinaryShape (zone) from SENSITIVE_AREAS
where db2gse.EnvelopesIntersect(zone, db2gse.PolyFromShape(cast(? as blob(1m)),
db2gse.coordref()..srid(0)))");
/* Prepare the SQL statement. */
SQLPrepare(hstmt, (UCHAR *)sqlstmt, SQL_NTS);
/* Set the pcbvalue1 length of the shape. */
pcbvalue1 = blob_len;
/* Bind the shape parameter */
SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_BINARY, SQL_BLOB, blob_len,
0, shape, blob_len, &pcbvalue1);
/* Execute the query */
rc = SQLExecute (hstmt);
/* Assign the results of the query (the Zone polygons) to the
   fetched_binary variable. */
SQLBindCol (hstmt, 1, SQL_C_Binary, fetched_binary, 100000, &ind_blob);
/* Fetch each polygon within the display window and display it. */
while(SQL_SUCCESS == (rc = SQLFetch(hstmt)))
  draw_polygon(fetched_binary);
```

GeometryFromShape

GeometryFromShape は形状と地理情報参照システム ID を引き数として、図形オブジェクトを戻します。

構文

db2gse.GeometryFromShape(ShapeGeometry Blob(1M), cr db2gse.coordref)

戻りタイプ

db2gse.ST_Geometry

例

以下の C コード断片には、 LOTS 表にデータを挿入する ODBC 関数 (DB2 地理情報エクステンダーの SQL 関数に組み込まれている) が含まれています。

作成された LOTS 表には 2 つの列があります。 LOT_ID 列はそれぞれの敷地を一意的に識別し、 LOT ポリゴン列にはそれぞれの敷地の図形が含まれています。

```
CREATE TABLE LOTS ( lot_id   integer,
                    lot      db2gse.ST_MultiPolygon);
```

GeometryFromShape 関数は形状を DB2 地理情報エクステンダー図形に変換します。 INSERT ステートメント全体は shp_sql に複写されます。 INSERT ステートメントには、 LOT_ID および LOT データを動的に受け入れるパラメーター・マーカーが含まれています。

```
/* Create the SQL insert statement to populate the lot id and the
   lot multipolygon. The question marks are parameter markers that
   indicate the lot_id and lot values that will be retrieved at
   runtime. */
strcpy (shp_sql,"insert into LOTS (lot_id, lot) values (?, db2gse.GeometryFromShape
(cast(? as blob(1m)), db2gse.coordref()..srid(0)))");
/* Allocate memory for the SQL statement handle and associate the
   statement handle with the connection handle. */
rc = SQLAllocStmt (handle, &hstmt);
/* Prepare the SQL statement for execution. */
rc = SQLPrepare (hstmt, (unsigned char *)shp_sql, SQL_NTS);
/* Bind the integer key value to the first parameter. */
pcbvalue1 = 0;
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_SLONG,
     SQL_INTEGER, 0, 0, &lot_id, 0, &pcbvalue1);
/* Bind the shape to the second parameter. */
pcbvalue2 = blob_len;
rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_BINARY,
     SQL_BLOB, blob_len, 0, shape_blob, blob_len, &pcbvalue2);
/* Execute the insert statement. */
rc = SQLExecute (hstmt);
```

EnvelopesIntersect

EnvelopesIntersect は、2 つの図形エンベロープが交差する場合に 1 (TRUE) を戻します。そうでない場合 0 (FALSE) を戻します。

構文

db2gse.EnvelopesIntersect(g1 db2gse.ST_Geometry, g2 db2gse.ST_Geometry)

戻りタイプ

整数

例

get_window 関数は、アプリケーションから表示ウィンドウの座標を検索します。 window パラメーターは、実際には、表示ポリゴンを表す座標のストリングを含む、ポリゴンの形状構造です。 PolyFromShape 関数は、表示ウィンドウの形状を DB2 地理情報エクステンダーのポリゴンに変換します。
EnvelopesIntersect 関数は DB2 地理情報エクステンダーのポリゴンを交差エンベロープとして使います。表示ウィンドウの内部または境界と交差するすべての SENSITIVE_AREAS ゾーン・ポリゴンが戻されます。それぞれのポリゴンは結果セットから取り出されて、draw_polygon 関数に渡されます。

```
/* Get the display window coordinates as a polygon shape.
get_window(&window)
/* Create the SQL expression. The db2gse.EnvelopesIntersect function
  will be used to limit the result set to only those zone polygons
  that intersect the envelope of the display window. */
strcpy(sqlstmt, "select db2gse.AsBinaryShape(zone) from SENSITIVE_AREAS where
db2gse.EnvelopesIntersect (zone, db2gse.PolyFromShape(cast(? as blob(1m)),
db2gse.coordref()..srid(0)))");
/* Set blob_len to the byte length of a 5 point shape polygon. */
blob_len = 128;
/* Prepare the SQL statement. */
SQLPrepare(hstmt, (UCHAR *)sqlstmt, SQL_NTS);
/* Set the pcbvalue1 to the window shape */
pcbvalue1 = blob_len;
/* Bind the shape parameter */
SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_BINARY, SQL_BLOB, blob_len,
0, window, blob_len, &pcbvalue1);
/* Execute the query */
rc = SQLExecute (hstmt);
/* Assign the results of the query, (the Zone polygons) to the
   fetched_binary variable. */
SQLBindCol (hstmt, 1, SQL_C_Binary, fetched_binary, 100000, &ind_blob);
/* Fetch each polygon within the display window and display it. */
while(SQL_SUCCESS == (rc = SQLFetch(hstmt))
  draw_polygon(fetched_binary);
```

Is3d

Is3d は形状オブジェクトを引き数とし、そのオブジェクトに 3D 座標があれば 1 (TRUE) を戻します。そうでなければ 0 (FALSE) を戻します。

構文

db2gse.Is3d(g db2gse.ST_Geometry)

戻りタイプ

整数

例

以下の CREATE TABLE ステートメントによって THREED_TEST 表が作成されます。この表には、整数タイプの GID 列と G1 図形列の 2 つの列があります。

```
CREATE TABLE THREED_TEST (gid smallint, g1 db2gse.ST_Geometry)
```

INSERT ステートメントによって 2 つのポイントが THREED_TEST 表に挿入されます。最初のポイントには Z 座標がありませんが、2 番目のポイントにはあります。

```
INSERT INTO THREED_TEST
VALUES(1, db2gse.ST_PointFromText('point (10 10)', db2gse.coordref()..srid(0)))
INSERT INTO THREED_TEST
VALUES (2, db2gse.ST_PointFromText('point z (10.92 10.12 5)',
db2gse.coordref()..srid(0)))
```

以下の SELECT ステートメントによって、GID 列の内容が Is3d 関数の結果とともにリストされます。この関数は、Z 座標のない最初の行に対して 0 を戻し、 Z 座標のある 2 番目の行に対して 1 を戻します。

```
SELECT gid, db2gse.Is3d (g1) "Is it 3d?" FROM THREED_TEST
```

以下の結果セットが戻されます。

```
gid      Is it 3d?
------   ----------
     1           0
     2           1
```

IsMeasured

IsMeasured は図形オブジェクトを引き数とし、そのオブジェクトに測定値があれば 1 (TRUE) を戻します。そうでなければ 0 (FALSE) を戻します。

構文

db2gse.IsMeasured(g db2gse.ST_Geometry)

戻りタイプ

整数

例

以下の CREATE TABLE ステートメントによって MEASURE_TEST 表が作成されます。この表には 2 つの列があります。 GID 列は行を一意的に識別し、G1 列にはポイント図形が格納されます。

```
CREATE TABLE MEASURE_TEST (gid smallint, g1 db2gse.ST_Geometry)
```

以下の INSERT ステートメントによって 2 つのレコードが MEASURE_TEST 表に挿入されます。最初のレコードは測定値を持たないポイントを格納します。 2 番目のレコードのポイントには測定値があります。

```
INSERT INTO MEASURE_TEST
VALUES(1, db2gse.ST_PointFromText('point (10 10)', db2gse.coordref()..srid(0)))
INSERT INTO MEASURE_TEST
VALUES (2, db2gse.ST_PointFromText('point m (10.92 10.12 5)', db2gse.coordref()..srid(0)))
```

以下の SELECT ステートメントおよび対応する結果セットによって、 GID 列および IsMeasured 関数の出力が表示されます。最初の行では、ポイントに測定値がないので、IsMeasured 関数は 0 を戻します。 2 番目の行では、ポイントに測定値があるので、1 を戻します。

```
SELECT gid, db2gse.IsMeasured (g1) "Has measures?" FROM MEASURE_TEST
gid      Has measures
------   ----------
     1            0
     2            1
```

LineFromShape

LineFromShape はポイント型の輪郭および地理情報参照システム ID を引き数として、折れ線を戻します。

構文

db2gse.Line FromShape(ShapeLineString Blob(1M), cr db2gse.coordref)

戻りタイプ

db2gse.ST_LineString

例

以下のコード断片では、SEWERLINES 表に、それぞれの下水路の固有の ID、サイズ・クラス、および図形を入れます。

以下の CREATE TABLE ステートメントによって SEWERLINES 表が作成されます。この表には 3 つの列があります。最初の列である SEWER_ID は、それぞれの下水路を一意的に識別します。 2 番目の列である CLASS は整数タイプで、下水路のタイプを識別します。通常は下水路の容量と関連付けられています。 3 番目の列である SEWER は折れ線タイプで、下水路の図形を格納します。

```
CREATE TABLE SEWERLINES (sewer_id   integer, class  integer,
                         sewer db2gse.ST_LineString);
/* Create the SQL insert statement to populate the sewer_id, size class and
   the sewer linestring. The question marks are parameter markers that
   indicate the sewer_id, class and sewer geometry values that will be
   retrieved at runtime. */
strcpy (shp_sql,"insert into sewerlines (sewer_id,class,sewer)
values (?,?, db2gse.Line FromShape (cast(? as blob(1m)),
db2gse.coordref()..srid(0)))");
/* Allocate memory for the SQL statement handle and associate the
   statement handle with the connection handle. */
rc = SQLAllocStmt (handle, &hstmt);
/* Prepare the SQL statement for execution. */
rc = SQLPrepare (hstmt, (unsigned char *)shp_sql, SQL_NTS);
/* Bind the integer key value to the first parameter. */
pcbvalue1 = 0;
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_SLONG,
   SQL_INTEGER, 0, 0, &sewer_id, 0, &pcbvalue1);
/* Bind the integer class value to the second parameter. */
pcbvalue2 = 0;
rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_SLONG,
   SQL_INTEGER, 0, 0, &sewer_class, 0, &pcbvalue2);;
/* Bind the shape to the third parameter. */
pcbvalue3 = blob_len;
```

```
rc = SQLBindParameter (hstmt, 3, SQL_PARAM_INPUT, SQL_C_BINARY,
     SQL_BLOB, blob_len, 0, sewer_shape, blob_len, &pcbvalue3);
/* Execute the insert statement. */
rc = SQLExecute (hstmt);
```

LocateAlong

LocateAlong は図形オブジェクトと測定値を引き数とし、測定値で見つかったポイントの集合を複数ポイントとして戻します。

構文

db2gse.LocateAlong(g db2gse.ST_Geometry, adistance Double)

戻りタイプ

db2gse.ST_Geometry

例

以下の CREATE TABLE ステートメントによって LOCATEALONG_TEST 表が作成されます。 LOCATEALONG_TEST には 2 つの列があります。 GID 列はそれぞれの行を一意的に識別し、G1 図形列にサンプルの図形が格納されます。

```
CREATE TABLE LOCATEALONG_TEST (gid integer, g1 db2gse.ST_Geometry)
```

以下の INSERT ステートメントによって 2 つの行が挿入されます。最初の行は複数折れ線であり、2 番目は複数ポイントです。

```
INSERT INTO db2gse.LOCATEALONG_TEST VALUES(
1, db2gse.ST_MLineFromText('multilinestring m ((10.29 19.23 5,23.82 20.29 6,30.19 18.47
                                    7,45.98 20.74 8),
                                  (23.82 20.29 6,30.98 23.98 7,42.92 25.98 8))',
                            db2gse.coordref()..srid(0)))
INSERT INTO db2gse.LocateAlong_TEST VALUES(
2, db2gse.ST_MPointFromText('multipoint m (10.29 19.23 5,23.82 20.29 6,30.19 18.47 7,
45.98 20.74 8,23.82 20.29 6,30.98 23.98 7,42.92 25.98)',
                            db2gse.coordref()..srid(0)))
```

以下の SELECT ステートメントおよび対応する結果セットでは、 LocateAlong 関数が、測定値 6.5 のポイントを見つけるように指示されます。最初の行は 2 つのポイントを含んでいる複数ポイントを戻します。しかし、2 番目の行は空のポイントを戻します。 (次元が 0 より大きい図形の) 線形機能の場合、LocateAlong がポイントを書き入れることがあります。しかし、複数ポイントの場合は、目標の測定値が正確に一致する必要があります。

```
SELECT gid, CAST(db2gse.ST_AsText(db2gse.LocateAlong (g1,6.5)) AS
varchar(96)) "Geometry"
FROM LOCATEALONG_TEST
GID         Geometry
----------- ----------------------------------------------------------------
          1 MULTIPOINT M ( 27.01000000 19.38000000 6.50000000, 27.40000000
22.14000000 6.50000000)
          2 POINT EMPTY
  2 record(s) selected.
```

以下の SELECT ステートメントおよび対応する結果セットでは、 LocateAlong 関数が両方の行に対して複数ポイントを戻します。目標測定値の 7 は、複数折れ線と複数ポイントの両方のソース・データの測定値に一致します。

```
SELECT gid,CAST(db2gse.ST_AsText(db2gse.LocateAlong (g1,7)) AS varchar(96)) "Geometry"
FROM LOCATEALONG_TEST
GID         Geometry
----------- ----------------------------------------------------------------
          1 MULTIPOINT M ( 30.19000000 18.47000000 7.00000000, 30.98000000
 23.98000000 7.00000000)
          2 MULTIPOINT M ( 30.19000000 18.47000000 7.00000000, 30.98000000
 23.98000000 7.00000000)
  2 record(s) selected.
```

LocateBetween

LocateBetween は 2 つの図形オブジェクトと 2 つの測定位置を引き数とし、図形を戻します。この図形は、2 つの測定位置の間にある切断されたパスのセットを表しています。

構文

db2gse.LocateBetween(g db2gse.ST_Geometry, adistance Double, anotherdistance Double)

戻りタイプ

db2gse.ST_Geometry

例

以下の CREATE TABLE ステートメントによって LOCATEBETWEEN_TEST 表が作成されます。 LOCATEBETWEEN_TEST には 2 つの列があります。GID 列はそれぞれの行を一意的に識別し、 G1 複数折れ線列にはサンプルの図形が格納されます。

```
CREATE TABLE LOCATEBETWEEN_TEST (gid integer, g1 db2gse.ST_Geometry)
```

以下の INSERT ステートメントによって 2 つの行が LOCATEBETWEEN_TEST 表に挿入されます。最初の行は複数折れ線であり、2 番目は複数ポイントです。

```
INSERT INTO db2gse.LOCATEBETWEEN_TEST
VALUES(1,db2gse.ST_MLineFromText('multilinestring m ((10.29 19.23 5,23.82 20.29 6,
                                          30.19 18.47 7,45.98 20.74 8),
                                         (23.82 20.29 6,30.98 23.98 7,
                                          42.92 25.98 8))',
                                 db2gse.coordref()..srid(0)))
INSERT INTO db2gse.LOCATEBETWEEN_TEST
VALUES(2, db2gse.ST_MPointFromText('multipoint m (10.29 19.23 5,23.82 20.29 6,
30.19 18.47 7,45.98 20.74 8,23.82 20.29 6,
30.98 23.98 7,42.92 25.98 8)',
                                 db2gse.coordref()..srid(0)))
```

以下の SELECT ステートメントおよび対応する結果セットによって、LocateBetween 関数が測定値 6.5 と測定値 7.5 の間 (境界を含む) にある測定値を見付ける方法が示されています。最初の行は、複数の折れ線を含んでいる複数折れ線を戻します。 2 番目の行は、ソース・データが複数ポイントであったため、複数ポイントを戻します。ソース・データの次元が 0 (ポイントまたは複数ポイント) の時は、完全に一致することが必要です。

```
SELECT gid, CAST(db2gse.ST_AsText(db2gse.LocateBetween (g1,6.5,7.5))
       AS varchar(96)) "Geometry"
FROM LOCATEBETWEEN_TEST
GID         Geometry
```

```
---------- ------------------------------------------------------------------
         1 MULTILINESTRING M ( 27.01000000 19.38000000 6.50000000, 31.19000000
18.47000000 7.00000000,38.09000000 19.61000000 7.50000000),(27.40000000 22.1400
0000 6.50000000, 30.98000000 23.98000000 7.00000000,36.95000000 24.98000000 7.5
0000000)
         2 MULTIPOINT M ( 30.19000000 18.47000000 7.00000000, 30.98000000 23.9
8000000 7.00000000)
  2 record(s) selected.
```

M

M はポイントを引き数とし、その測定値を戻します。

構文

db2gse.M(p db2gse.ST_Point)

戻りタイプ

倍精度

例

以下の CREATE TABLE ステートメントによって M_TEST 表が作成されます。 M_TEST には 2 つの列があります。 GID 整数列は行を一意的に識別し、PT1 ポイント列にはサンプルの図形が格納されます。

```
CREATE TABLE M_TEST (gid integer, pt1 db2gse.ST_Point)
```

以下の INSERT ステートメントによって、測定値のあるポイントを含んでいる行と、測定値のないポイントを含んでいる行が挿入されます。

```
INSERT INTO db2gse.M_TEST
VALUES(1, db2gse.ST_PointFromText('point (10.02 20.01)', db2gse.coordref()..srid(0)))
INSERT INTO db2gse.M_TEST
VALUES(2, db2gse.ST_PointFromText('point  zm(10.02 20.01 5.0 7.0)',
db2gse.coordref()..srid(0)))
```

以下の M ステートメントおよび対応する結果セットでは、 M 関数によってポイントの測定値の値がリストされます。最初のポイントには測定値がないので、M 関数は NULL を戻します。

```
SELECT gid, db2gse.M (pt1) "The measure" FROM M_TEST
GID         The measure
----------- ------------------------
          1                         -
          2   +7.00000000000000E+000
  2 record(s) selected.
```

MLine FromShape

MLineFromShape は複数折れ線型の形状と地理情報参照システム ID を引き数とし、複数折れ線を戻します。

構文

db2gse.MLineFromShape(ShapeMultiLineString Blob(1M), cr db2gse.coordref)

戻りタイプ

db2gse.ST_MultiLineString

例

以下のコード断片では、WATERWAYS 表に、固有の id、name、および water 複数折れ線を入れます。

WATERWAYS 表が作成されます。この表には、表に格納されているそれぞれの河川とその支流を識別する ID 列と NAME 列があります。河川とその支流は複数の折れ線の集合体となっていることが多いので、WATER 列は複数折れ線です。

```
CREATE TABLE WATERWAYS (id        integer,
                        name      varchar(128),
                        water     db2gse.ST_MultiLineString);
/* Create the SQL insert statement to populate the id, name and
   multilinestring. The question marks are parameter markers that
   indicate the id, name and water values that will be retrieved at
   runtime. */
strcpy (shp_sql,"insert into WATERWAYS (id,name,water)
values (?,?, db2gse.MLineFromShape (cast(? as blob(1m)),
db2gse.coordref()..srid(0)))");
/* Allocate memory for the SQL statement handle and associate the
   statement handle with the connection handle. */
rc = SQLAllocStmt (handle, &hstmt);
/* Prepare the SQL statement for execution. */
rc = SQLPrepare (hstmt, (unsigned char *)shp_sql, SQL_NTS);
/* Bind the integer id value to the first parameter. */
pcbvalue1 = 0;
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_SLONG,
   SQL_INTEGER, 0, 0, &id, 0, &pcbvalue1);
/* Bind the varchar name value to the second parameter. */
pcbvalue2 = name_len;
rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR,
     SQL_CHAR, name_len, 0, &name, name_len, &pcbvalue2);
/* Bind the shape to the third parameter. */
pcbvalue3 = blob_len;
rc = SQLBindParameter (hstmt, 3, SQL_PARAM_INPUT, SQL_C_BINARY,
     SQL_BLOB, blob_len, 0, water_shape, blob_len, &pcbvalue3);
/* Execute the insert statement. */
rc = SQLExecute (hstmt);
```

MPointFromShape

MPointFromShape は、複数ポイント型の形状と地理情報参照システム ID を引き数とし、複数ポイントを戻します。

構文

db2gse.MPointFromShape(ShapeMultiPoint (1M), srs db2gse.coordref)

戻りタイプ

db2gse.ST_MultiPoint

例

以下のコード断片では、生物学者の SPECIES_SITINGS 表にデータを入れます。

3 つの列を持つ SPECIES_SITINGS 表が作成されます。 species 列および genus 列はそれぞれの行を一意的に識別し、 sitings 複数ポイント列には種類ごとの生息地域が格納されます。

```
CREATE TABLE SPECIES_SITINGS (species  varchar(32),
                              genus varchar(32),
                              sitings  db2gse.ST_MultiPoint);
/* Create the SQL insert statement to populate the species, genus and
   sitings. The question marks are parameter markers that indicate the
   name and water values that will be retrieved at runtime. */
strcpy (shp_sql,"insert into SPECIES_SITINGS (species,genus,sitings)
values (?,?, db2gse.MPointFromShape (cast(? as blob(1m)),
db2gse.coordref()..srid(0)))");
/* Allocate memory for the SQL statement handle and associate the
   statement handle with the connection handle. */
rc = SQLAllocStmt (handle, &hstmt);
/* Prepare the SQL statement for execution. */
rc = SQLPrepare (hstmt, (unsigned char *)shp_sql, SQL_NTS);
/* Bind the varchar species value to the first parameter. */
pcbvalue1 = species_len;
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR,
     SQL_CHAR, species_len, 0, species, species_len, &pcbvalue1);
/* Bind the varchar genus value to the second parameter. */
pcbvalue2 = genus_len;
rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR,
   SQL_CHAR, genus_len, 0, name, genus_len, &pcbvalue2);
/* Bind the shape to the third parameter. */
pcbvalue3 = blob_len;
rc = SQLBindParameter (hstmt, 3, SQL_PARAM_INPUT, SQL_C_BINARY,
     SQL_BLOB, sitings_len, 0, sitings_shape, sitings_len, &pcbvalue3);
/* Execute the insert statement. */
rc = SQLExecute (hstmt);
```

MPolyFromShape

MPolyFromShape は、複数ポリゴン型の形状と地理情報参照システム ID を引き数とし、複数ポリゴンを戻します。

構文

db2gse.MPolyFromShape(ShapeMultiPolygon Blob(1m), srs db2gse.coordref)

戻りタイプ

db2gse.ST_MultiPolygon

例

以下のコード断片では、LOTS 表にデータを入れます。

LOTS 表には、それぞれの敷地を一意的に識別する lot_id、および敷地境界線の図形を含む複数ポリゴンが格納されています。

```
CREATE TABLE LOTS (  lot_id integer, lot db2gse.ST_MultiPolygon );
/* Create the SQL insert statement to populate the lot_id and lot. The
   question marks are parameter markers that indicate the lot_id and lot
   values that will be retrieved at runtime. */
strcpy (shp_sql,"insert into LOTS (lot_id,lot)
values (?, db2gse.MPolyFromShape (cast(? as blob(1m)),
db2gse.coordref()..srid(0)))");
/* Allocate memory for the SQL statement handle and associate the
   statement handle with the connection handle. */
rc = SQLAllocStmt (handle, &hstmt);
/* Prepare the SQL statement for execution. */
rc = SQLPrepare (hstmt, (unsigned char *)shp_sql, SQL_NTS);
/* Bind the lot_id integer value to the first parameter. */
pcbvalue1 = 0;
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_INTEGER,
     SQL_INTEGER, 0, 0, &lot_id, 0, &pcbvalue1);
/* Bind the lot shape to the second parameter. */
pcbvalue2 = lot_len;
rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_BINARY,
     SQL_BLOB, lot_len, 0, lot_shape, lot_len, &pcbvalue2);
/* Execute the insert statement. */
rc = SQLExecute (hstmt);
```

PointFromShape

PointFromShape はポイント型の輪郭と地理情報参照システム ID を引き数とし、ポイントを戻します。

構文

db2gse.PointFromShape(db2gse.ShapePoint blob(1M), srs db2gse.coordref)

戻りタイプ

db2gse.ST_Point

例

以下のプログラム断片では、HAZARDOUS_SITES 表にデータを入れます。

危険地帯は、以下の CREATE TABLE ステートメントを用いて作成される HAZARDOUS_SITES 表に格納されます。ポイントとして定義された location 列には、それぞれの危険地帯の地理上の中心の位置が格納されます。

```
CREATE TABLE HAZARDOUS_SITES (site_id   integer,
                              name      varchar(128),
                              location  db2gse.ST_Point);
/* Create the SQL insert statement to populate the site_id, name and
   location. The question marks are parameter markers that indicate the
   site_id, name and location values that will be retrieved at runtime. */
strcpy (shp_sql,"insert into HAZARDOUS_SITES (site_id, name, location)
values (?,?, db2gse.PointFromShape (cast(? as blob(1m)), db2gse.coordref()..srid(0)))");
/* Allocate memory for the SQL statement handle and associate the
   statement handle with the connection handle. */
rc = SQLAllocStmt (handle, &hstmt);
/* Prepare the SQL statement for execution. */
rc = SQLPrepare (hstmt, (unsigned char *)shp_sql, SQL_NTS);
/* Bind the site_id integer value to the first parameter. */
pcbvalue1 = 0;
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_INTEGER,
     SQL_INTEGER, 0, 0, &site_id, 0, &pcbvalue1);
/* Bind the name varchar value to the second parameter. */
pcbvalue2 = name_len;
rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR,
   SQL_CHAR, 0, 0, name, 0, &pcbvalue2);
/* Bind the location shape to the third parameter. */
pcbvalue3 = location_len;
rc = SQLBindParameter (hstmt, 3, SQL_PARAM_INPUT, SQL_C_BINARY,
     SQL_BLOB, location_len, 0, location_shape, location_len, &pcbvalue3);
/* Execute the insert statement. */
rc = SQLExecute (hstmt);
```

PolyFromShape

PolyFromShape はポリゴン型の形状と地理情報参照システム ID を引き数とし、ポリゴンを戻します。

構文

db2gse.PolyFromShape (ShapePolygon Blob(1M), srs db2gse.coordref)

戻りタイプ

db2gse.ST_Polygon

例

以下のプログラム断片では、SENSITIVE_AREAS 表にデータを入れます。疑問符 (?) は、実行時に検索される ID、名前、サイズ、タイプおよびゾーン値に対するパラメーター・マーカーを表しています。

SENSITIVE_AREAS 表には、汚染の恐れがある公共施設について記述する複数の列と、それらの公共施設のポリゴン図形を格納する zone 列が含まれています。

```
CREATE TABLE SENSITIVE_AREAS (id        integer,
                              name      varchar(128),
                              size      float,
                              type      varchar(10),
                              zone  db2gse.ST_Polygon);
/* Create the SQL insert statement to populate the id, name, size, type and
   zone. The question marks are parameter markers that indicate the
   id, name, size, type and zone values that will be retrieved at runtime. */
strcpy (shp_sql,"insert into SENSITIVE_AREAS (id, name, size, type, zone)
values (?,?,?,?, db2gse.PolyFromShape (cast(? as blob(1m)),
db2gse.coordref()..srid(0)))");
/* Allocate memory for the SQL statement handle and associate the
   statement handle with the connection handle. */
rc = SQLAllocStmt (handle, &hstmt);
/* Prepare the SQL statement for execution. */
rc = SQLPrepare (hstmt, (unsigned char *)shp_sql, SQL_NTS);
/* Bind the id integer value to the first parameter. */
pcbvalue1 = 0;
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_INTEGER,
     SQL_INTEGER, 0, 0, &site_id, 0, &pcbvalue1);
/* Bind the name varchar value to the second parameter. */
pcbvalue2 = name_len;
rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR,
   SQL_CHAR, 0, 0, name, 0, &pcbvalue2);
/* Bind the size float to the third parameter. */
pcbvalue3 = 0;
rc = SQLBindParameter (hstmt, 3, SQL_PARAM_INPUT, SQL_C_FLOAT,
     SQL_REAL, 0, 0, &size, 0, &pcbvalue3);
/* Bind the type varchar to the fourth parameter. */
```

```
pcbvalue4 = type_len;
rc = SQLBindParameter (hstmt, 4, SQL_PARAM_INPUT, SQL_C_CHAR,
     SQL_VARCHAR, type_len, 0, type, type_len, &pcbvalue4);
/* Bind the zone polygon to the fifth parameter. */
pcbvalue5 = zone_len;
rc = SQLBindParameter (hstmt, 5, SQL_PARAM_INPUT, SQL_C_BINARY,
     SQL_BLOB, zone_len, 0, zone_shp, zone_len, &pcbvalue5);
/* Execute the insert statement. */
rc = SQLExecute (hstmt);
```

ShapeToSQL

ShapeToSQL は、事前に割り当てられたバイナリー表現で db2gse.ST_Geometry 値を構成します。 SRID 値 0 が自動的に使用されます。

構文

db2gse.ST_ShapeToSQL(ShapeGeometry blob(1M))

戻りタイプ

db2gse.ST_Geometry

例

以下の C コード断片には、 LOTS 表にデータを挿入する ODBC 関数 (DB2 地理情報エクステンダーの SQL 関数に組み込まれている) が含まれています。作成された LOTS 表には 2 つの列があります。 lot_id 列はそれぞれの敷地を一意的に識別し、 lot 複数ポリゴン列にはそれぞれの敷地の図形が含まれています。

```
CREATE TABLE lots (lot_id  integer,
                   lot      db2gse.ST_MultiPolygon);
```

ShapeToSQL 関数は形状を DB2 地理情報エクステンダー図形に変換します。 INSERT ステートメント全体は shp_sql に複写されます。 INSERT ステートメントには、 LOT_id および lot データを動的に受け入れるパラメーター・マーカーが含まれています。

```
/* Create the SQL insert statement to populate the lot id and the
   lot multipolygon. The question marks are parameter markers that
   indicate the lot_id and lot values that will be retrieved at
   run time. */
strcpy (shp_sql,"insert into lots (lot_id, lot) values(?,
db2gse.ShapeToSQL(cast(? as blob(1m))))");
/* Allocate memory for the SQL statement handle and associate the
   statement handle with the connection handle. */
rc = SQLAllocStmt (handle, &hstmt);
/* Prepare the SQL statement for execution. */
rc = SQLPrepare (hstmt, (unsigned char *)shp_sql, SQL_NTS);
/* Bind the integer key value to the first parameter. */
pcbvalue1 = 0;
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_SLONG,
     SQL_INTEGER, 0, 0, &lot_id, 0, &pcbvalue1);
/* Bind the shape to the second parameter. */
pcbvalue2 = blob_len;
rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_BINARY,
     SQL_BLOB, blob_len, 0, shape_blob, blob_len, &pcbvalue2);
/* Execute the insert statement. */
rc = SQLExecute (hstmt);
```

ST_Area

ST_Area はポリゴンまたは複数ポリゴンを引き数とし、面積を戻します。

構文

db2gse.ST_Area(s db2gse.ST_Surface)

戻りタイプ

倍精度

例

都市計画担当者は建築面積のリストを必要としています。このリストを入手するために、 GIS 技術者は建物のフットプリントごとに建物 ID と面積を選択します。

建物のフットプリントは、以下のような CREATE TABLE ステートメントを用いて作成された BUILDINGFOOTPRINTS 表の中に収められています。

```
CREATE TABLE BUILDINGFOOTPRINTS (   building_id integer,
                                    lot_id      integer,
                                    footprint   db2gse.ST_MultiPolygon);
```

都市計画担当者の要求を満たすために、技術担当者は BUILDINGFOOTPRINTS の中から固有キー、 building_id (建物 ID)、およびそれぞれの建物のフットプリントの面積を使用します。

```
SELECT building_id, db2gse.ST_Area (footprint) "Area"
FROM BUILDINGFOOTPRINTS;
```

SELECT ステートメントによって、以下の結果セットが戻されます。

```
building_id   Area
------------  ------------------------
         506    +1.40768000000000E+003
        1208    +2.55759000000000E+003
         543    +1.80786000000000E+003
         178    +2.08671000000000E+003
         .
         .
         .
```

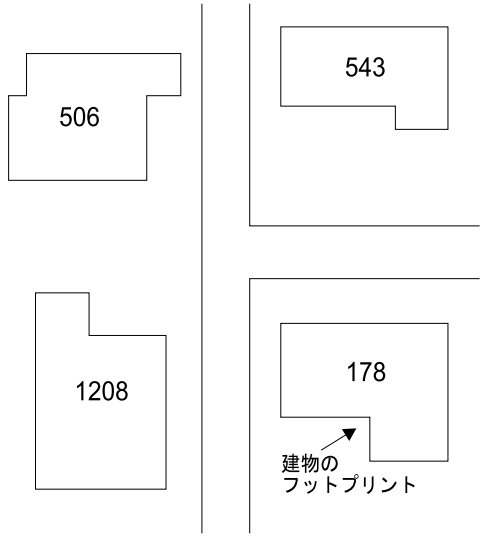


図 28. 面積による建物のフットプリントの検索. 建物 ID 番号が付いた 4 つの建物のフットプリントが、隣接する道路とともに表示されています。

ST_AsBinary

ST_AsBinary は図形オブジェクトを引き数とし、事前に割り当てられたバイナリー表現を戻します。

構文

db2gse.ST_AsBinary(g db2gse.ST_Geometry)

戻りタイプ

BLOB(1m)

例

下記のコード断片は、 ST_AsBinary 関数が BUILDINGFOOTPRINTS 表のフットプリントの複数ポリゴンを WKB 複数ポリゴンに変換する様子を示しています。これらの複数ポリゴンは、このアプリケーションの draw_polygon 関数に渡されて表示されます。

```
/* Create the SQL expression. */
strcpy(sqlstmt, "select db2gse.ST_AsBinary (footprint) from BUILDINGFOOTPRINTS
where db2gse.EnvelopesIntersect(footprint, db2gse.ST_PolyFromWKB(cast(? as blob(1m)),
db2gse.coordref()..srid(0)))");
/* Prepare the SQL statement. */
SQLPrepare(hstmt, (UCHAR *)sqlstmt, SQL_NTS);
/* Set the pcbvalue1 length of the shape. */
pcbvalue1 = blob_len;
/* Bind the shape parameter */
SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_BINARY, SQL_BLOB, blob_len,
0, shape, blob_len, &pcbvalue1);
/* Execute the query */
rc = SQLExecute (hstmt);
/* Assign the results of the query (the Zone polygons) to the
  fetched_binary variable.
 */
SQLBindCol (hstmt, 1, SQL_C_Binary, fetched_binary, 100000, &ind_blob);
/* Fetch each polygon within the display window and display it. */
while(SQL_SUCCESS == (rc = SQLFetch(hstmt)))
  draw_polygon(fetched_binary);
```

ST_AsText

db2gse.ST_AsText は図形オブジェクトを引き数とし、事前に割り当てられたテキスト表現を戻します。

構文

db2gse.ST_AsText(g db2gse.ST_Geometry)

戻りタイプ

Varchar(4000)

例

以下のシナリオで、db2gse.ST_AsText 関数は、 HAZARDOUS_SITES のロケーション・ポイントをテキスト記述に変換します。

```
CREATE TABLE HAZARDOUS_SITES (site_id   integer,
                              name      varchar(40),
                              location  db2gse.ST_Point);
INSERT INTO HAZARDOUS_SITES
VALUES (102,
        'W. H. Kleenare Chemical Repository',
        db2gse.ST_PointFromText('point (1020.12 324.02)', db2gse.coordref()..srid(0)));
SELECT site_id, name, cast(db2gse.ST_AsText(location) as varchar(40)) "Location"
FROM HAZARDOUS_SITES;
```

SELECT ステートメントによって、以下の結果セットが戻されます。

```
SITE_ID Name                                Location
------- ----------------------------------- ----------------------------------------
    102 W. H. Kleenare Chemical Repository  POINT (1020.00000000 324.00000000)
```

ST_Boundary

ST_Boundary は図形オブジェクトを引き数とし、結合された境界を図形オブジェクトとして戻します。

構文

db2gse.ST_Boundary(g db2gse.ST_Geometry)

戻りタイプ

db2gse.ST_Geometry

例

下記のコード断片では、BOUNDARY_TEST という名前の表が作成されます。BOUNDARY_TEST には 2 つの列があります。 GEOTYPE は varchar として定義され、G1 はスーパークラスの図形として定義されています。それに続く INSERT ステートメントでは、それぞれのサブクラス図形を挿入します。ST_Boundary 関数は、G1 図形列に格納されている各サブクラスの境界を検索します。結果として戻される図形の次元は、入力された図形の次元よりも常に 1 つ少ないことに注意してください。ポイントおよび複数ポイントの場合、結果は常に空の図形オブジェクト (次元 1) である境界となります。折れ線および複数折れ線の場合は、複数ポイント (次元 0) が戻されます。ポリゴンまたは複数ポリゴンの場合は、常に複数折れ線 (次元 1) が戻されます。

```
CREATE TABLE BOUNDARY_TEST (GEOTYPE varchar(20), G1 db2gse.ST_Geometry)
INSERT INTO BOUNDARY_TEST
VALUES('Point',
       db2gse.ST_PointFromText('point (10.02 20.01)',
                               db2gse.coordref()..srid(0)))
INSERT INTO BOUNDARY_TEST
VALUES ('Linestring',
       db2gse.ST_LineFromText('linestring  (10.02 20.01,10.32 23.98,11.92 25.64)',
                               db2gse.coordref()..srid(0)))
INSERT INTO BOUNDARY_TEST
VALUES('Polygon',
       db2gse.ST_PolyFromText('polygon  ((10.02 20.01,11.92 35.64,25.02 34.15,
                                    19.15 33.94, 10.02 20.01))',
                               db2gse.coordref()..srid(0)))
INSERT INTO BOUNDARY_TEST
VALUES('Multipoint',
       db2gse.ST_MPointFromText('multipoint (10.02 20.01,10.32 23.98,11.92 25.64)',
                               db2gse.coordref()..srid(0)))
INSERT INTO BOUNDARY_TEST
VALUES('Multilinestring',
       db2gse.ST_MLineFromText('multilinestring ((10.02 20.01,10.32 23.98,11.92 25.64),
                                        (9.55 23.75,15.36 30.11))',
                               db2gse.coordref()..srid(0)))
INSERT INTO BOUNDARY_TEST
VALUES('Multipolygon',
db2gse.ST_MPolyFromText('multipolygon (((10.02 20.01,11.92 35.64,25.02 34.15,
                                          19.15 33.94,10.02 20.01)),
                                        ((51.71 21.73,73.36 27.04,71.52 32.87,
                                          52.43 31.90,51.71 21.73)))',
                               db2gse.coordref()..srid(0)))
```

```
SELECT GEOTYPE,
       CAST(db2gse.ST_AsText(db2gse.ST_Boundary (G1)) as varchar(280)) "The boundary"
FROM BOUNDARY_TEST
GEOTYPE              The boundary
-------------------- ------------------------------------------------------------
Point                POINT EMPTY
Linestring           MULTIPOINT ( 10.02000000 20.01000000, 11.92000000
 25.64000000)
Polygon              MULTILINESTRING (( 10.02000000 20.01000000, 19.15000000
 33.94000000, 25.02000000 34.15000000, 11.92000000 35.64000000, 10.02000000
 20.01000000))
Multipoint           POINT EMPTY
Multilinestring      MULTIPOINT ( 9.55000000 23.75000000, 10.02000000
 20.01000000, 11.92000000 25.64000000, 15.36000000 30.11000000)
Multipolygon         MULTILINESTRING (( 51.71000000 21.73000000, 73.36000000
 27.04000000, 71.52000000 32.87000000, 52.43000000 31.90000000, 51.71000000
 21.73000000),( 10.02000000 20.01000000, 19.15000000 33.94000000, 25.02000000
 34.15000000, 11.92000000 35.64000000, 10.02000000 20.01000000))
6 record(s) selected.
```

ST_Buffer

ST_Buffer は図形オブジェクトと距離を引き数とし、ソース・オブジェクトを囲む図形オブジェクトを戻します。

構文

db2gse.ST_Buffer(g db2gse.ST_Geometry , adistance Double)

戻りタイプ

db2gse.ST_Geometry

例

自治体の行政責任者が、半径 5 マイルが学校、病院、養護施設などの重要地域と重なっている危険地帯のリストを必要としています。重要地域は、以下の CREATE TABLE ステートメントを用いて作成される SENSITIVE_AREAS 表に格納されます。 ZONE 列はポリゴンとして定義され、それぞれの重要地域の輪郭として格納されます。

```
CREATE TABLE SENSITIVE_AREAS (id        integer,
                              name      varchar(128),
                              size      float,
                              type      varchar(10),
                              zone  db2gse.ST_Polygon);
```

危険地帯は、以下の CREATE TABLE ステートメントを用いて作成される HAZARDOUS_SITES 表に格納されます。ポイントとして定義された LOCATION 列には、それぞれの危険地帯の地理上の中心の位置が格納されます。

```
CREATE TABLE HAZARDOUS_SITES (site_id   integer,
                              name      varchar(128),
                              location  db2gse.ST_Point);
```

SENSITIVE_AREAS 表および HAZARDOUS_SITES 表は、 db2gse.ST_Overlaps 関数によって結合されます。この関数は、SENSITIVE_AREAS 行のうち、zone ポリゴンが HAZARDOUS_SITES location ポイントから半径 5 マイルの緩衝地帯と重なっているすべての行について 1 (TRUE) を戻します。

```
SELECT sa.name "Sensitive Areas", hs.name "Hazardous Sites"
FROM SENSITIVE_AREAS sa, HAZARDOUS_SITES hs
WHERE db2gse.ST_Overlaps(sa.zone, db2gse.ST_Buffer (hs.location,(5 * 5280))) = 1;
```

196ページの図29 では、この行政地域内のいくつかの重要地区が、危険地帯の位置から半径 5 マイルの緩衝地帯の中にあります。 2 つの半径 5 マイルの緩

衝地帯のうち、両方が病院を含み、1 つが学校を含んでいます。ただし、養護施設は 2 つの半径の外側の安全な場所にあります。

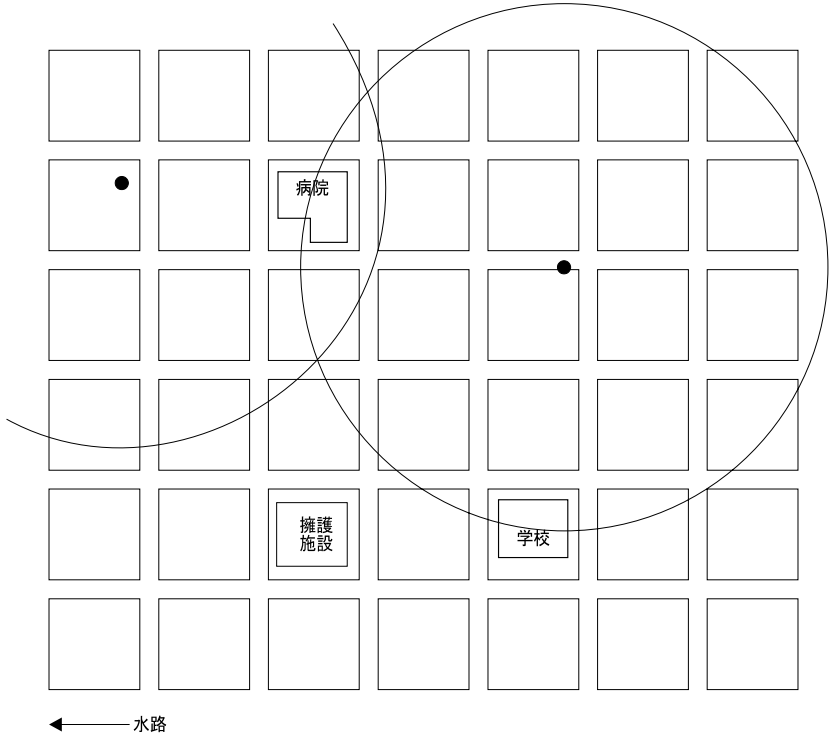


図 29. 半径 5 マイルの緩衝地帯がポイントに適用されている

ST_Centroid

ST_Centroid はポリゴンまたは複数ポリゴンを引き数とし、図形的な中心をポイントとして戻します。

構文

db2gse.ST_Centroid(s db2gse.ST_Surface)
db2gse.ST_Centroid(ms db2gse.ST_MultiSurface)

戻りタイプ

表面の場合: db2gse.ST_Point

例

市の GIS 技術担当者が、建物のフットプリントの複数ポリゴンを、建物密度図で単一のポイントとして表示しようとしています。

建物のフットプリントは、以下のような CREATE TABLE ステートメントを用いて作成された BUILDINGFOOTPRINTS 表の中に収められています。

```
CREATE TABLE BUILDINGFOOTPRINTS (building_id integer,
                                 lot_id      integer,
                                 footprint   db2gse.ST_MultiPolygon);
```

ST_Centroid 関数は、それぞれの建物のフットプリントの複数ポリゴンの中心を戻します。 AsBinaryShape 関数は、中心ポイントを、アプリケーションで認識される外部表現である形状に変換します。

```
SELECT building_id,
       CAST(db2gse.AsBinaryShape(db2gse.ST_Centroid (footprint)) as blob(1m))
"Centroid"
FROM BUILDINGFOOTPRINTS;
```

ST_Contains

ST_Contains は 2 つの形状オブジェクトを引き数とし、最初のオブジェクトが 2 番目のオブジェクトを完全に含んでいる場合に 1 (TRUE) を戻します。そうでなければ 0 (FALSE) を戻します。

構文

db2gse.ST_Contains(g1 db2gse.ST_Geometry, g2 db2gse.ST_Geometry)

戻りタイプ

整数

例

以下の例では、2 つの表が作成されます。 1 つの表には都市内の建物のフットプリントが含まれ、もう 1 つの表にはその建物の敷地が含まれています。都市計画担当者が、すべての建物のフットプリントが完全に敷地内に収まっていることを確かめたいとします。

どちらの表でも、複数ポリゴンのデータ・タイプに建物のフットプリントの図形と敷地の図形が格納されています。データベース設計者は、両方について複数ポリゴンを選択しました。これは、河川のような地形によって敷地が分断されている場合があることと、建物のフットプリントが複数の建物から構成されている場合があることに気付いたためです。

```
CREATE TABLE BUILDINGFOOTPRINTS (building_id integer,
                                 lot_id      integer,
                                 footprint   db2gse.ST_MultiPolygon);
CREATE TABLE LOTS (lot_id integer, lot db2gse.ST_MultiPolygon);
```

都市計画の担当者は、1 つの敷地に完全には収まっていない建物を最初に選びます。

```
SELECT building_id
FROM BUILDINGFOOTPRINTS, LOTS
WHERE db2gse.ST_Contains(lot,footprint) = 0;
```

都市計画の担当者は、最初の照会によって、敷地ポリゴンの外側にフットプリントのあるすべての建物識別コードのリストが戻されることを理解しています。しかし、この情報からは、ほかの建物に正しい敷地識別コードが割り当てられているかどうかが分からないことも理解しています。以下に示す 2 番目の照会は、 BUILDINGFOOTPRINTS 表の lot_id 列に対して、データ保全性の検査を実行します。

```
SELECT bf.building_id "building id", bf.lot_id "buildings lot_id",
       LOTS.lot_id "LOTS lot_id"
FROM BUILDINGFOOTPRINTS bf, LOTS
WHERE db2gse.ST_Contains(lot,footprint) = 1 AND LOTS.lot_id <> bf.lot_id;
```

図30 では、建物 ID の付いた建物のフットプリントが、敷地内に収まっています。敷地の境界線は、点線で示されています。ここには示されていませんが、これらの線は道路の中心線にまで伸びており、敷地とその中のフットプリントを完全に囲んでいます。

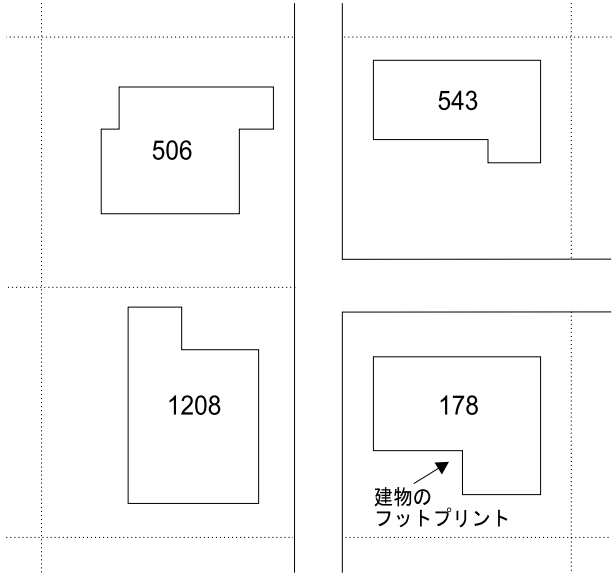


図 30. ST_Contains を使用して、すべての建物がその敷地内に収まっていることを確認する

ST_ConvexHull

ST_ConvexHull は図形オブジェクトを引き数とし、凸 包 (convex hull) を戻します。

構文

db2gse.ST_ConvexHull(g db2gse.ST_Geometry)

戻りタイプ

db2gse.ST_Geometry

例

この例では、2 つの列 (GEOTYPE および G1) がある CONVEXHULL_TEST 表が作成されます。 GEOTYPE 列は varchar(20) であり、 G1 に格納されている図形のサブクラス名を格納します。 g1 は図形として定義されています。

```
CREATE TABLE CONVEXHULL_TEST (geotype varchar(20), g1 db2gse.ST_Geometry)
```

次の INSERT ステートメントによって、それぞれのサブクラス・タイプの図形が CONVEXHULL_TEST 表に挿入されます。

```
INSERT INTO CONVEXHULL_TEST
VALUES('Point',
db2gse.ST_PointFromText('point (10.02 20.01)', db2gse.coordref()..srid(0)))
INSERT INTO CONVEXHULL_TEST
VALUES ('Linestring',
       db2gse.ST_LineFromText('linestring  (10.02 20.01,10.32 23.98,11.92 25.64)',
                              db2gse.coordref()..srid(0)))
INSERT INTO CONVEXHULL_TEST
VALUES('Polygon',
       db2gse.ST_PolyFromText('polygon  ((10.02 20.01,11.92 35.64,25.02 34.15,
                                          19.15 33.94,10.02 20.01))',
                              db2gse.coordref()..srid(0)))
INSERT INTO CONVEXHULL_TEST
VALUES('Multipoint',
       db2gse.ST_MPointFromText('multipoint (10.02 20.01,10.32 23.98,11.92 25.64)',
                              db2gse.coordref()..srid(0)))
INSERT INTO CONVEXHULL_TEST
VALUES('Multilinestring',
       db2gse.ST_MLineFromText('multilinestring ((10.02 20.01,10.32 23.98,11.92 25.64),
                                       (9.55 23.75,15.36 30.11))',
                              db2gse.coordref()..srid(0)))
INSERT INTO CONVEXHULL_TEST
VALUES('Multipolygon',
db2gse.ST_MPolyFromText('multipolygon (((10.02 20.01,11.92 35.64,25.02 34.15,
                                         19.15 33.94,10.02 20.01)),
                                       ((51.71 21.73,73.36 27.04,71.52 32.87,
                                         52.43 31.90,51.71 21.73)))',
                              db2gse.coordref()..srid(0)))
```

以下の SELECT ステートメントよって、 GEOTYPE 列に格納されているサブクラス名および 凸 包がリストされます。 ST_ConvexHull 関数によって生成さ

れる 凸 包は、ST_AsText 関数によってテキストに変換されます。その後、varchar(256) にキャストされます。これは、ST_AsText のデフォルト出力が varchar(4000) であるためです。

```
SELECT GEOTYPE, CAST(db2gse.ST_AsText(db2gse.ST_ConvexHull(G1))) as varchar(256)
"The convexhull"
FROM CONVEXHULL_TEST
```

ST_CoordDim

ST_CoordDim は、ST_Geometry 値の座標の次元を戻します。座標の次元については、 139ページの『ポイント』を参照してください。

構文

db2gse.ST_CoordDim(g1 db2gse.ST_Geometry)

戻りタイプ

整数

例

geotype 列および g1 列がある coorddim_test 表が作成されます。 geotype 列には、g1 図形列に格納されている図形サブクラスの名前が格納されます。

```
CREATE TABLE coorddim_test (geotype varchar(20), g1 db2gse.ST_Geometry)
```

INSERT ステートメントによって、サンプルのサブクラスが coorddim_test 表に挿入されます。

```
INSERT INTO coorddim_test VALUES(
    'Point', db2gse.ST_PointFromText('point (10.02 20.01)', db2gse.coordref()..srid(0))
)
INSERT INTO coorddim_test VALUES(
'Linestring',
       db2gse.ST_LineFromText('linestring  (10.02 20.01,10.32 23.98,11.92 25.64)',
db2gse.coordref()..srid(0))
)
INSERT INTO coorddim_test VALUES(
'Polygon', db2gse.ST_PolyFromText('polygon ((10.02 20.01,11.92 35.64,25.02 34.15,
19.15 33.94,10.02 20.01))', db2gse.coordref()..srid(0))
)
INSERT INTO coorddim_test VALUES(
'Multipoint', db2gse.ST_MPointFromText('multipoint (10.02 20.01,10.32 23.98,
11.92 25.64)', db2gse.coordref()..srid(0))
)
INSERT INTO coorddim_test VALUES(
'Multilinestring', db2gse.ST_MLineFromText('multilinestring ((10.02 20.01,
10.32 23.98,11.92 25.64),(9.55 23.75,15.36 30.11))', db2gse.coordref()..srid(0))
)
INSERT INTO coorddim_test VALUES(
'Multipolygon',
       MPolyFromText('multipolygon (((10.02 20.01,11.92 35.64,25.02 34.15,
19.15 33.94,10.02 20.01)),((51.71 21.73,73.36 27.04,71.52 32.87,
52.43 31.90,51.71 21.73)))', db2gse.coordref()..srid(0))
)
```

SELECT ステートメントによって、 geotype の座標次元とともに geotype 列に格納されているサブクラス名がリストされます。

```
SELECT geotype, db2gse.ST_coordDim(g1)' coordinate_dimension'
FROM coorddim_test
GEOTYPE        coordinate_dimension
------------------- -------------------------
ST_Point                                   2
```

```
ST_Linestring                           2
ST_Polygon              2
ST_Multipoint                           2
ST_Multilinestring                      2
ST_Multipolygon         2
6 record(s) selected.
```

ST_Crosses

ST_Crosses は 2 つの図形オブジェクトを引き数とし、それらの交差部分が、ソース・オブジェクトの最大次元よりも 1 つ小さい次元の図形オブジェクトとなる場合に、 1 (TRUE) を戻します。交差オブジェクトには、両方のソース図形の中にあり、なおかつどちらのソース・オブジェクトとも等しくないポイントが含まれます。そうでない場合、0 (FALSE) を戻します。

構文

db2gse.ST_Crosses(g1 db2gse.ST_Geometry, g2 db2gse.ST_Geometry)

戻りタイプ

整数

例

地方自治体は、自治体内のあらゆる水路から 5 マイル以内に危険な廃棄物貯蔵施設を建設することを禁じた、新しい条例の制定を検討しています。自治体の GIS 担当者は、河川および支流の正確な表現を所有しています。この表示は、WATERWAYS 表に複数折れ線として格納されています。ただし、それぞれの危険な廃棄物処理施設について、単一のポイント location を所有しているだけです。

```
CREATE TABLE WATERWAYS (id        integer,
                        name      varchar(128),
                        water      db2gse.ST_MultiLineString);
CREATE TABLE HAZARDOUS_SITES ( site_id   integer,
                               name      varchar(128),
                               location  db2gse.ST_Point);
```

提案されている条例に違反する公共施設が存在することを自治体の行政責任者に知らせなければならないかどうか判断するため、 GIS 管理担当者は危険地帯の緩衝地帯を設定して、バッファー・ポリゴン内を河川が横切っているかどうか確認する必要があります。 ST_Crosses 述部は、緩衝地帯付きの HAZARDOUS_SITES を WATERWAYS と比較します。このため、自治体の条例案による半径を横切る水路のレコードだけが戻されます。

```
SELECT ww.name "River or stream", hs.name "Hazardous site"
FROM WATERWAYS ww, HAZARDOUS_SITES hs
WHERE db2gse.ST_Crosses(db2gse.ST_Buffer(hs.location,(5 * 5280)),ww.water) = 1;
```

205ページの図31 の中で、危険な廃棄物処理施設から 5 マイルの緩衝地帯は、自治体の行政地域を流れる河川網と重なっています。この河川網は複数折れ線

として定義されています。このため、結果セットには、この半径を横切るセグメントの一部であるすべての折れ線が含まれます。

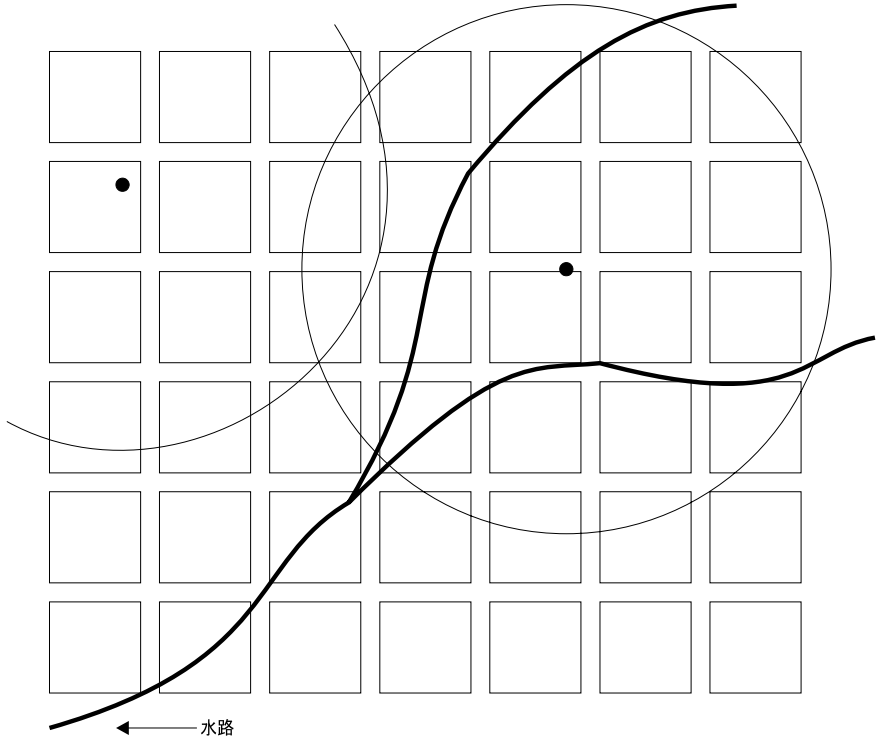


図 31. ST_Crosses を用いて、危険な廃棄物区域を通過する水系を見つける

ST_Difference

ST_Difference は 2 つの図形オブジェクトを引き数とし、ソース・オブジェクト間の差である図形オブジェクトを戻します。

構文

db2gse.ST_Difference(g1 db2gse.ST_Geometry, g2 db2gse.ST_Geometry)

戻りタイプ

db2gse.ST_Geometry

例

都市開発担当者は、市内の敷地区域のうち建物の建っていない区域の面積の合計を知っていなければなりません。したがって、建物の建っている区域を除いた敷地区域の面積の合計が必要となります。

```
CREATE TABLE BUILDINGFOOTPRINTS (building_id integer,
                                 lot_id      integer,
                                 footprint   db2gse.ST_MultiPolygon);
CREATE TABLE LOTS (lot_id   integer,
                   lot      db2gse.ST_MultiPolygon);
```

都市開発担当者は、lot_id について BUILDINGFOOTPRINTS 表および LOTS 表を等価結合します。その後、建物のフットプリントを除いた、敷地の差となる区域の面積の合計を計算します。

```
SELECT SUM(db2gse.ST_Area(db2gse.ST_Difference(lot,footprint)))
FROM BUILDINGFOOTPRINTS bf, LOTS
WHERE bf.lot_id = LOTS.lot_id;
```

ST_Dimension

ST_Dimension は図形オブジェクトを引き数とし、その次元を戻します。

構文

db2gse.ST_Dimension(g1 db2gse.ST_Geometry)

戻りタイプ

整数

例

GEOTYPE 列および G1 列がある DIMENSION_TEST 表が作成されます。GEOTYPE 列には、G1 図形列に格納されている図形サブクラスの名前が格納されます。

```
CREATE TABLE DIMENSION_TEST (geotype varchar(20), g1 db2gse.ST_Geometry)
```

INSERT ステートメントによって、サンプルのサブクラスが DIMENSION_TEST 表に挿入されます。

```
INSERT INTO DIMENSION_TEST
VALUES('Point',
db2gse.ST_PointFromText('point (10.02 20.01)', db2gse.coordref()..srid(0)))
INSERT INTO DIMENSION_TEST
VALUES ('Linestring',
       db2gse.ST_LineFromText('linestring  (10.02 20.01,10.32 23.98,11.92 25.64)',
                              db2gse.coordref()..srid(0)))
INSERT INTO DIMENSION_TEST
VALUES('Polygon',
       db2gse.ST_PolyFromText('polygon  ((10.02 20.01,11.92 35.64,25.02 34.15,
                                          19.15 33.94,10.02 20.01))',
                              db2gse.coordref()..srid(0)))
INSERT INTO DIMENSION_TEST
VALUES('Multipoint',
       db2gse.ST_MPointFromText('multipoint (10.02 20.01,10.32 23.98,11.92 25.64)',
                              db2gse.coordref()..srid(0)))
INSERT INTO DIMENSION_TEST
VALUES('Multilinestring',
       db2gse.ST_MLineFromText('multilinestring ((10.02 20.01,10.32 23.98,11.92 25.64),
                                        (9.55 23.75,15.36 30.11))',
                              db2gse.coordref()..srid(0)))
INSERT INTO DIMENSION_TEST
VALUES('Multipolygon',
db2gse.ST_MPolyFromText('multipolygon (((10.02 20.01,11.92 35.64,25.02 34.15,
                                         19.15 33.94,10.02 20.01)),
                                        ((51.71 21.73,73.36 27.04,71.52 32.87,
                                         52.43 31.90,51.71 21.73)))',
                    db2gse.coordref)..srid(0)))
```

以下の SELECT ステートメントによって、 geotype の次元とともに GEOTYPE 列に格納されているサブクラス名がリストされます。

```
SELECT geotype, db2gse.ST_Dimension(g1) "The dimension"
FROM DIMENSION_TEST
```

以下の結果セットが戻されます。

```
GEOTYPE              The dimension
-------------------- -------------
ST_Point                       0
ST_Linestring                  1
ST_Polygon                     2
ST_Multipoint                  0
ST_Multilinestring             1
ST_Multipolygon                2
6 record(s) selected.
```

ST_Disjoint

ST_Disjoint は 2 つの図形を引き数とし、 2 つの図形の交差が空のセットを生成する場合に 1 (TRUE) を戻し、そうでない場合に 0 (FALSE) を戻します。

構文

db2gse.ST_Disjoint(g1 db2gse.ST_Geometry, g2 db2gse.ST_Geometry)

戻りタイプ

整数

例

ある保険会社で、保険の対象となる町の病院、養護施設、および学校を決定する必要があります。この手順には、それぞれの公共施設に対する危険な廃棄物施設の影響がどの程度かを判別することが含まれます。この保険会社では、汚染される危険のない公共施設だけを保険の対象として考慮しようと考えています。保険会社に依頼された GIS コンサルタントは、危険な廃棄物施設から半径 5 マイル以内にないすべての公共施設を特定するという任務を与えられています。

SENSITIVE_AREAS 表には、汚染の恐れがある公共施設について記述する複数の列と、それらの公共施設のポリゴン図形を格納する ZONE 列が含まれています。

```
CREATE TABLE SENSITIVE_AREAS (id        integer,
                              name      varchar(128),
                              size      float,
                              type      varchar(10),
                              zone      db2gse.ST_Polygon);
```

HAZARDOUS_SITES 表には、地点を識別する SITE_ID 列と NAME 列が格納されています。それぞれの地点の実際の地理上の位置は、LOCATION 列に格納されています。

```
CREATE TABLE HAZARDOUS_SITES (site_id   integer,
                              name      varchar(128),
                              location  db2gse.ST_Point);
```

以下の SELECT ステートメントによって、危険な廃棄物の区域から半径 5 マイル以内にない重要地域の名前がすべてリストされます。関数の結果が 1 ではなく 0 に等しく設定されている場合は、この照会では、ST_Intersects 関数で ST_Disjoint 関数を置き換えることもできます。これは、ST_Intersects と ST_Disjoint が正反対の結果を戻すためです。

```
SELECT sa.name
FROM SENSITIVE_AREAS sa, HAZARDOUS_SITES hs
WHERE db2gse.ST_Disjoint(db2gse.ST_Buffer(hs.location,(5 * 5280)),sa.zone) = 1;
```

図32 では、重要地域が危険な廃棄物地域の半径 5 マイルと比較されています。 ST_Disjoint 関数が 1 (TRUE) を戻す重要地域は、養護施設だけです。 2 つの図形がまったく交差しない場合、ST_Disjoint 関数は常に 1 を戻します。

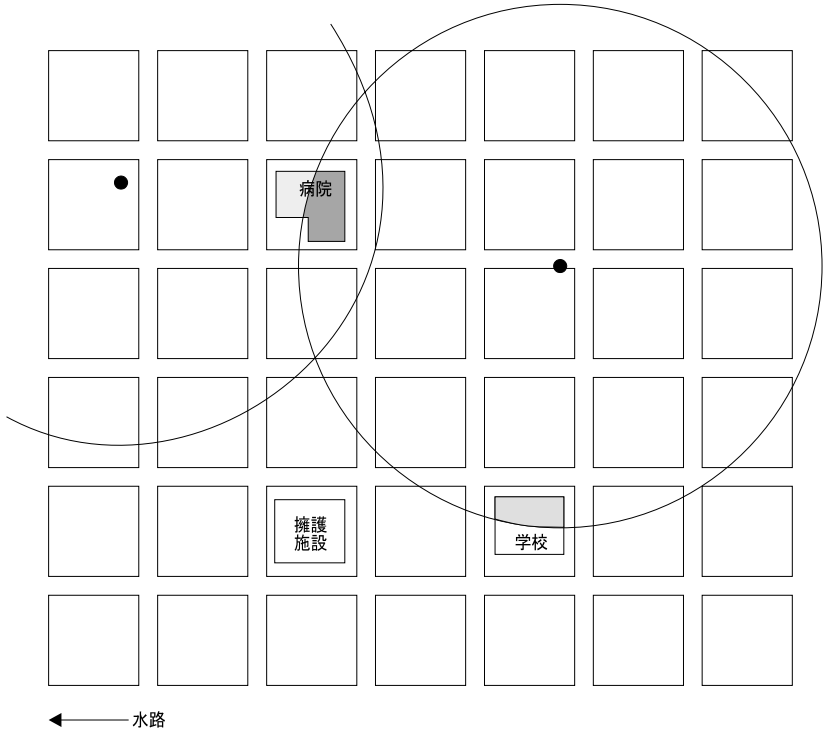


図 32. ST_Disjoint を用いて、(交差する) 危険な廃棄物施設の地域にない建物を見つける

ST_Distance

ST_Distance は 2 つの図形を引き数とし、両者の間の最短距離を戻します。

構文

db2gse.ST_Distance(g1 db2gse.ST_Geometry, g2 db2gse.ST_Geometry)

戻りタイプ

倍精度

例

都市計画担当者は、敷地境界線から 1 フィート以内にあるすべての建物のリストを必要としています。

BUILDINGFOOTPRINTS 表の BUILDING_ID 列は、それぞれの建物を一意的に識別しています。 LOT_ID 列は、それぞれの建物が属する敷地を識別しています。 footprint 複数ポリゴンには、それぞれの建物の図形が格納されています。

```
CREATE TABLE BUILDINGFOOTPRINTS (  building_id integer,
                                   lot_id      integer,
                                   footprint   db2gse.ST_MultiPolygon);
```

LOTS 表には、それぞれの敷地を一意的に識別する敷地識別子、および敷地境界線の図形を含む複数ポリゴンが格納されています。

```
CREATE TABLE LOTS ( lot_id   integer,
                        lot   db2gse.ST_MultiPolygon);
```

この照会は、敷地境界線から 1 フィート以内にある建物 ID のリストを戻します。 ST_Distance 関数は、footprint と lot 複数ポリゴンとの地理情報上の結合を実行します。ただし、bf.lot_id と LOTS.lot_id との等価結合により、必ず同じ敷地にある複数ポリゴンだけが ST_Distance 関数によって比較されます。

```
SELECT bf.building_id
  FROM BUILDINGFOOTPRINTS bf, LOTS
 WHERE bf.lot_id = LOTS.lot_id AND
       db2gse.ST_Distance(bf.footprint, db2gse.ST_Boundary(LOTS.lot)) <= 1.0;
```

ST_Endpoint

ST_Endpoint は折れ線を引き数とし、その折れ線の最後のポイントを戻します。

構文

db2gse.ST_Endpoint(c db2gse.ST_Curve)

戻りタイプ

db2gse.ST_Point

例

ENDPOINT_TEST 表には、GID 整数列が格納されています。 GID 整数列は、それぞれの行と、折れ線を収める LN1 折れ線列を一意的に識別します。

```
CREATE TABLE ENDPOINT_TEST (gid integer, ln1 db2gse.ST_LineString)
```

INSERT ステートメントによって、折れ線が ENDPOINT_TEST 表に挿入されます。最初の INSERT ステートメントには Z 座標や測定値がありませんが、2 番目のステートメントにはあります。

```
INSERT INTO ENDPOINT_TEST
VALUES( 1,
       db2gse.ST_LineFromText('linestring (10.02 20.01,23.73 21.92,30.10 40.23)',
                              db2gse.coordref()..srid(0)))
INSERT INTO ENDPOINT_TEST
VALUES (2,
       db2gse.ST_LineFromText('linestring zm (10.02 20.01 5.0 7.0,23.73 21.92 6.5 7.1,
                                    30.10 40.23 6.9 7.2)',
                              db2gse.coordref()..srid(0)))
```

以下の SELECT ステートメントによって、 GID 列とともに、ST_Endpoint 関数の出力がリストされます。 ST_Endpoint 関数は、 ST_AsText 関数によってテキストに変換されるポイント図形を生成します。 CAST 関数は、ST_AsText 関数のデフォルトの varchar(4000) 値を varchar(60) に短縮するのに使用されています。

```
SELECT gid, CAST(db2gse.ST_AsText(db2gse.ST_Endpoint(ln1)) AS varchar(60)) "Endpoint"
FROM ENDPOINT_TEST
```

以下の結果セットが戻されます。

```
GID         Endpoint
----------- ------------------------------------------------------------
          1 POINT ( 30.10000000 40.23000000)
          2 POINT ZM ( 30.10000000 40.23000000 7.00000000 7.20000000)
  2 record(s) selected.
```

ST_Envelope

ST_Envelope は 1 つの図形オブジェクトを引き数とし、それを囲むボックスを図形として戻します。

構文

db2gse.ST_Envelope(g db2gse.ST_Geometry)

戻りタイプ

db2gse.ST_Geometry

例

ENVELOPE_TEST 表の GEOTYPE 列には、 G1 図形列に格納されている図形サブクラスの名前が格納されます。

```
CREATE TABLE ENVELOPE_TEST (geotype varchar(20), g1 db2gse.ST_Geometry)
```

以下の INSERT ステートメントによって、それぞれの図形サブクラスが ENVELOPE_TEST 表に挿入されます。

```
INSERT INTO ENVELOPE_TEST
VALUES('Point',
       db2gse.ST_PointFromText('point (10.02 20.01)', db2gse.coordref()..srid(0)))
INSERT INTO ENVELOPE_TEST
VALUES ('Linestring',
        db2gse.ST_LineFromText('linestring (10.01 20.01, 10.01 30.01, 10.01 40.01)',
        db2gse.coordref()..srid(0)))
INSERT INTO ENVELOPE_TEST
VALUES ('Linestring',
       db2gse.ST_LineFromText('linestring  (10.02 20.01,10.32 23.98,11.92 25.64)',
       db2gse.coordref()..srid(0)))
INSERT INTO ENVELOPE_TEST
VALUES('Polygon',
       db2gse.ST_PolyFromText('polygon  ((10.02 20.01,11.92 35.64,25.02 34.15,
                                          19.15 33.94,10.02 20.01))',
                              db2gse.coordref()..srid(0)))
INSERT INTO ENVELOPE_TEST
VALUES('Multipoint',
       db2gse.ST_MPointFromText('multipoint (10.02 20.01,10.32 23.98,11.92 25.64)',
                              db2gse.coordref()..srid(0)))
INSERT INTO ENVELOPE_TEST
VALUES('Multilinestring',
       db2gse.ST_MLineFromText('multilinestring ((10.01 20.01,20.01 20.01,30.01 20.01),
                                        (30.01 20.01,40.01 20.01,50.01 20.01))',
       db2gse.coordref()..srid(0)))
INSERT INTO ENVELOPE_TEST
VALUES('Multilinestring',
       db2gse.ST_MLineFromText('multilinestring ((10.02 20.01,10.32 23.98,11.92 25.64),
                                        (9.55 23.75,15.36 30.11))',
                              db2gse.coordref()..srid(0)))
INSERT INTO ENVELOPE_TEST
VALUES('Multipolygon',
       db2gse.ST_MPolyFromText('multipolygon (((10.02 20.01,11.92 35.64,25.02 34.15,
                                          19.15 33.94,10.02 20.01)),
                                        ((51.71 21.73,73.36 27.04,71.52 32.87,
                                          52.43 31.90,51.71 21.73)))',
                              db2gse.coordref()..srid(0)))
```

以下の SELECT ステートメントによって、エンベロープの隣にサブクラス名がリストされます。 ST_Envelope 関数はポイント、折れ線、もしくはポリゴンのいずれかを戻すため、出力は ST_AsText 関数を用いてテキストに変換されます。 CAST 関数は、 ST_AsText 関数のデフォルトの varchar(4000) の結果を varchar(280) に変換します。

```
SELECT GEOTYPE, CAST(db2gse.ST_AsText(db2gse.ST_Envelope(g1)) AS varchar(280))
"The envelope"
FROM ENVELOPE_TEST
```

以下の結果セットが戻されます。

```
GEOTYPE              The envelope
-------------------- ------------------------------------------------------------
Point                POINT ( 10.02000000 20.01000000)
Linestring           LINESTRING ( 10.01000000 20.01000000, 10.01000000
 40.01000000)
Linestring           POLYGON (( 10.02000000 20.01000000, 11.92000000
 20.01000000, 11.92000000 25.64000000, 10.02000000 25.64000000, 10.02000000
 20.01000000))
Polygon              POLYGON (( 10.02000000 20.01000000, 25.02000000
 20.01000000, 25.02000000 35.64000000, 10.02000000 35.64000000, 10.02000000
 20.01000000))
Multipoint           POLYGON (( 10.02000000 20.01000000, 11.92000000
 20.01000000, 11.92000000 25.64000000, 10.02000000 25.64000000, 10.02000000
 20.01000000))
Multilinestring      LINESTRING ( 10.01000000 20.01000000, 50.01000000
 20.01000000)
Multilinestring      POLYGON (( 9.55000000 20.01000000, 15.36000000
 20.01000000, 15.36000000 30.11000000, 9.55000000 30.11000000, 9.55000000
 20.01000000))
Multipolygon         POLYGON (( 10.02000000 20.01000000, 73.36000000
 20.01000000, 73.36000000 35.64000000, 10.02000000 35.64000000, 10.02000000
 20.01000000))
  8 record(s) selected.
```

ST_Equals

ST_Equals は 2 つの図形を比較して、等しい図形であれば 1 (TRUE) を戻し、そうでなければ 0 (FALSE) を戻します。

構文

db2gse.ST_Equals(g1 db2gse.ST_Geometry, g2 db2gse.ST_Geometry)

戻りタイプ

整数

例

都市の GIS 技術担当者は、 BUILDINGFOOTPRINTS 表のデータの一部が何らかの原因で複製されたのではないかと考えています。技術担当者はこの表を照会して、フットプリントの複数ポリゴンの中で互いに等しいものがないか判別します。

BUILDINGFOOTPRINTS 表は、以下のステートメントを用いて作成されます。 BUILDING_ID 列は建物を一意的に識別し、LOT_ID は建物の敷地を識別します。 FOOTPRINT 列には、建物の図形が格納されます。

```
CREATE TABLE BUILDINGFOOTPRINTS ( building_id integer,
                                  lot_id      integer,
                                  footprint   db2gse.ST_MultiPolygon);
```

BUILDINGFOOTPRINTS 表は、ST_Equals 述部によって地理情報的に自身に結合されています。この述部は、2 つの等しい複数ポリゴンを発見すると 1 を戻します。 bf1.building_id <> bf2.building_id 条件は、同じ図形が比較されるのを防ぐために必要です。

```
SELECT bf1.building_id, bf2.building_id
FROM BUILDINGFOOTPRINTS bf1, BUILDINGFOOTPRINTS bf2
WHERE db2gse.ST_Equals(bf1.footprint,bf2.footprint) = 1
      and bf1.building_id <> bf2.building_id;
```

ST_ExteriorRing

ST_ExteriorRing はポリゴンを引き数とし、その外部リングを折れ線として戻します。

構文

db2gse.ST_ExteriorRing(s db2gse.ST_Polygon)

戻りタイプ

db2gse.ST_LineString

例

南洋諸島で鳥の群生の調査を計画しているある鳥類学者は、特定の鳥の採食地域が水際に限定されていることを知りました。島々の食料供給能力を計算するために、この学者は島の周囲の長さを知る必要があります。いくつかの池のある島もありますが、池の湖岸線はより攻撃的な別の種類の鳥に占領されています。このため、この鳥類学者は島々の外部リングの周囲の長さを知る必要があります。

ISLANDS 表の ID 列と NAME 列はそれぞれの島を識別し、タイプ ST_Polygon の LAND 列にはそれぞれの図形が格納されています。

```
CREATE TABLE ISLANDS (id    integer,
                      name  varchar(32),
                      land  db2gse.ST_Polygon);
```

ST_ExteriorRing 関数は、それぞれの島のポリゴンの外部リングを折れ線として抽出します。折れ線の長さは、length 関数によって算出されます。島々の折れ線の長さの合計は SUM 関数によって算出されます。

```
SELECT SUM(db2gse.ST_length(db2gse.ST_ExteriorRing (land))) FROM ISLANDS;
```

217ページの図33 で、島の外部リングはそれぞれの島と海との生態学的な境界を表しています。いくつかの島には湖があり、ポリゴンの内部リングとして表されています。

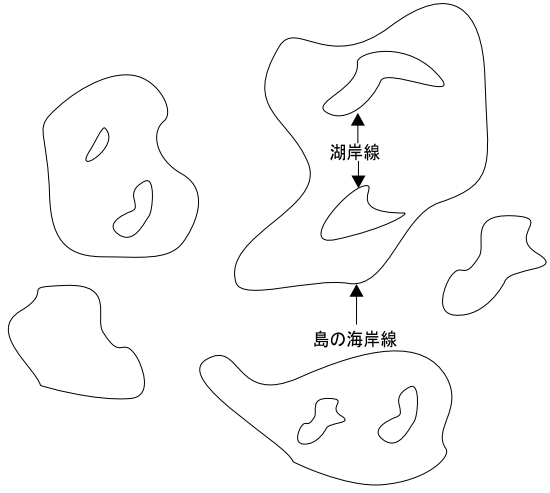


図 33. ST_ExteriorRing を用いて島の海岸線の長さを判別する

ST_GeometryFromText

ST_GeometryFromText は事前割り当てのテキスト表現と地理情報参照システム ID を引き数とし、図形オブジェクトを戻します。

構文

db2gse.ST_GeometryFromText(geometryTaggedText Varchar(4000), cr db2gse.coordref)

戻りタイプ

db2gse.ST_Geometry

例

GEOMETRY_TEST 表には GID 整数列が含まれ、それぞれの行と、図形が格納されている G1 列を一意的に識別しています。

```
CREATE TABLE GEOMETRY_TEST (gid smallint, g1 db2gse.ST_Geometry)
```

INSERT ステートメントは、GEOMETRY_TEST 表の GID 列および G1 列にデータを挿入します。 ST_GeometryFromText 関数は、それぞれの図形のテキスト表現を、対応する DB2 地理情報エクステンダーのインスタンス化可能なサブクラスに変換します。

```
INSERT INTO GEOMETRY_TEST
VALUES(1, db2gse.ST_GeometryFromText('point (10.02 20.01)', db2gse.coordref()..srid(0)))
INSERT INTO GEOMETRY_TEST
VALUES (2,
        db2gse.ST_GeometryFromText('linestring (10.01 20.01, 10.01 30.01, 10.01 40.01)',
                                   db2gse.coordref()..srid(0)))
INSERT INTO GEOMETRY_TEST
VALUES(3,
       db2gse.ST_GeometryFromText('polygon ((10.02 20.01,11.92 35.64,25.02 34.15,
                                             19.15 33.94,10.02 20.01))',
                                   db2gse.coordref()..srid(0)))
INSERT INTO GEOMETRY_TEST
VALUES(4,
       db2gse.ST_GeometryFromText('multipoint (10.02 20.01,10.32 23.98,11.92 25.64)',
                                   db2gse.coordref()..srid(0)))
INSERT INTO GEOMETRY_TEST
VALUES(5,
       db2gse.ST_GeometryFromText('multilinestring ((10.02 20.01,10.32 23.98,
                                                    11.92 25.64),
                                                   (9.55 23.75,15.36 30.11))',
                                   db2gse.coordref()..srid(0)))
INSERT INTO GEOMETRY_TEST
VALUES(6,
       db2gse.ST_GeometryFromText('multipolygon (((10.02 20.01,11.92 35.64,25.02 34.15,
                                                   19.15 33.94,10.02 20.01)),
                                                  ((51.71 21.73,73.36 27.04,71.52 32.87,
                                                    52.43 31.90,51.71 21.73)))',
                                   db2gse.coordref()..srid(0)))
```

ST_GeomFromWKB

ST_GeomFromWKB は事前割り当てのバイナリー表現と地理情報参照システム ID を引き数とし、図形オブジェクトを戻します。

構文

db2gse.ST_GeomFromWKB(WKBGeometry Blob(1M), cr db2gse.coordref)

戻りタイプ

db2gse.ST_Geometry

例

以下の C コード断片には、 LOTS 表にデータを挿入する ODBC 関数 (DB2 地理情報エクステンダーの SQL 関数に組み込まれている) が含まれています。

作成された LOTS 表には 2 つの列があります。 LOT_ID 列はそれぞれの敷地を一意的に識別し、 LOT 複数ポリゴン列にはそれぞれの敷地の図形が含まれています。

```
CREATE TABLE LOTS ( lot_id   integer,
                    lot      db2gse.ST_MultiPolygon);
```

ST_GeomFromWKB 関数は、WKB 表現を DB2 地理情報エクステンダーの図形に変換します。 INSERT ステートメント全体は wkb_sql char string に複写されます。 INSERT ステートメントには、 LOT_ID および LOT データを動的に受け入れるパラメーター・マーカーが含まれています。

```
/* Create the SQL insert statement to populate the lot id and the
   lot multipolygon. The question marks are parameter markers that
   indicate the lot_id and lot values that will be retrieved at
   runtime. */
strcpy (wkb_sql,"insert into LOTS (lot_id, lot) values (?, db2gse.ST_GeomFromWKB
(cast(? as blob(1m)), db2gse.coordref()..srid(0)))");
/* Allocate memory for the SQL statement handle and associate the
   statement handle with the connection handle. */
rc = SQLAllocStmt (handle, &hstmt);
/* Prepare the SQL statement for execution. */
rc = SQLPrepare (hstmt, (unsigned char *)wkb_sql, SQL_NTS);
/* Bind the integer key value to the first parameter. */
pcbvalue1 = 0;
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_SLONG,
     SQL_INTEGER, 0, 0, &lot_id, 0, &pcbvalue1);
/* Bind the shape to the second parameter. */
pcbvalue2 = blob_len;
```

```
rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_BINARY,
     SQL_BLOB, blob_len, 0, shape_blob, blob_len, &pcbvalue2);
/* Execute the insert statement. */
rc = SQLExecute (hstmt);
```

ST_GeometryN

ST_GeometryN はコレクションと整数インデックスを引き数とし、コレクションの中の *n* 番目の図形オブジェクトを戻します。

構文

db2gse.ST_GeometryN(g db2gse.ST_GeomCollection, n Integer)

戻りタイプ

db2gse.ST_Geometry

例

都市計画の担当者は、建物のフットプリント全体が敷地の複数ポリゴンの最初のポリゴン内にあるかどうかを調べる必要があります。

BUILDING_ID 列は、BUILDINGFOOTPRINTS 表のそれぞれの行を一意的に識別します。 LOT_ID 列は、それぞれの建物の敷地を識別します。 FOOTPRINT 列には建物の図形が格納されています。

```
CREATE TABLE BUILDINGFOOTPRINTS ( building_id integer,
                                  lot_id      integer,
                                  footprint   db2gse.ST_MultiPolygon);
CREATE TABLE LOTS ( lot_id   integer,
                    lot      db2gse.ST_MultiPolygon);
```

この照会は、BUILDINGFOOTPRINTS 内で最初の敷地ポリゴンの中にあるすべての建物フットプリントについて、 building_id および lot_id をリストします。 ST_GeometryN 関数は、multipolygon 配列にある最初の敷地ポリゴンを戻します。

```
SELECT bf.building_id,bf.lot_id
FROM BUILDINGFOOTPRINTS bf,LOTS
WHERE db2gse.ST_Within(footprint, db2gse.ST_GeometryN (lot,1)) = 1
      AND bf.lot_id = LOTS.lot_id;
```

ST_GeometryType

ST_GeometryType は ST_Geometry オブジェクトを引き数とし、その図形タイプをストリングとして戻します。

構文

db2gse.ST_GeometryType (g db2gse.ST_Geometry)

戻りタイプ

Varchar(4000)

例

GEOMETRYTYPE_TEST 表には G1 図形列が含まれています。

```
CREATE TABLE GEOMETRYTYPE_TEST(g1 db2gse.ST_Geometry)
```

以下の INSERT ステートメントによって、それぞれの図形サブクラスが G1 列に挿入されます。

```
INSERT INTO GEOMETRYTYPE_TEST
VALUES(db2gse.ST_GeometryFromText('point (10.02 20.01)', db2gse.coordref()..srid(0)))
INSERT INTO GEOMETRYTYPE_TEST
VALUES (db2gse.ST_GeometryFromText('linestring (10.01 20.01, 10.01 30.01, 10.01 40.01)',
                                   db2gse.coordref()..srid(0)))
INSERT INTO GEOMETRYTYPE_TEST
VALUES(db2gse.ST_Geometrytype_test values(db2gse.ST_GeomFromText('polygon ((10.02
20.01,11.92 35.64,25.02 34.15,19.15 33.94, 10.02 20.01))',
                                 db2gse.coordref()..srid(0))))
INSERT INTO GEOMETRYTYPE_TEST
VALUES(db2gse.ST_GeometryFromText('multipoint (10.02
 20.01,10.32 23.98,11.92 25.64)',
                                   db2gse.coordref()..srid(0)))
INSERT INTO GEOMETRYTYPE_TEST
VALUES(db2gse.ST_GeometryFromText('multilinestring ((10.02 20.01,10.32 23.98,
                                              11.92 25.64),
                                         (9.55 23.75,15.36 30.11))',
                                   db2gse.coordref()..srid(0)))
INSERT INTO GEOMETRYTYPE_TEST
VALUES(db2gse.ST_GeometryFromText('multipolygon (((10.02 20.01,11.92 35.64,25.02 34.15,
                                           19.15 33.94,10.02 20.01)),
                                          ((51.71 21.73,73.36 27.04,71.52 32.87,
                                            52.43 31.90,51.71 21.73)))',
                                   db2gse.coordref()..srid(0)))
```

以下の SELECT ステートメントは、 G1 図形列に格納されているそれぞれのサブクラスの図形タイプをリストします。

```
SELECT db2gse.ST_GeometryType(g1) "Geometry type" FROM GEOMETRYTYPE_TEST
```

以下の結果セットが戻されます。

```
Geometry type
--------------------------------
ST_Point
ST_LineString
```

```
ST_Polygon
ST_MultiPoint
ST_MultiLineString
ST_MultiPolygon
6 record(s) selected.
```

ST_InteriorRingN

ポリゴンの *n* 番目の内部リングを折れ線として戻します。リングは図形的方向付けによって編成されてはいません。内部の図形照合ルーチンで定義されている規則に従って編成されています。したがって、リングの順序を事前定義することはできません。

構文

ST_InteriorRingN(p ST_Polygon, n Integer)

戻りタイプ

db2gse.ST_LineString

例

南洋諸島で鳥の群生の調査を計画しているある鳥類学者は、特定の受動的な種類の鳥の採食地域は海岸線に限定されていることが分かっています。一部の島には湖が存在しています。そのような湖の湖岸は、より攻撃的な別の種類の鳥によって占領されています。それぞれの島について、湖の周囲の長さの合計がある一定の限界値を超えると、攻撃的な鳥の数が増えすぎて海岸に生息する受動的な種類の鳥の存続が脅かされることを、この鳥類学者は知っています。このため、この鳥類学者は島々の内部リングの周囲の長さの総和を知る必要があります。

225ページの図34 で、島の外部リングはそれぞれの島と海との生態学的な境界を表しています。いくつかの島には湖があり、ポリゴンの内部リングとして表されています。

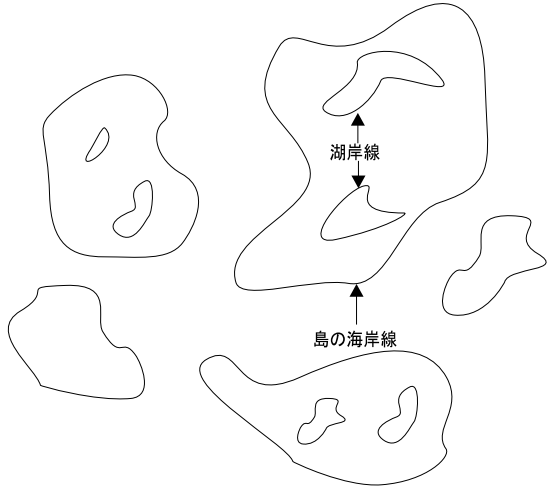


図 34. ST_InteriorRingN を用いて、それぞれの島の湖岸線の長さを判別する

ISLANDS 表の ID 列と name 列はそれぞれの島を識別し、 land ポリゴン列にはその島の図形が格納されています。

```
CREATE TABLE ISLANDS (id    integer,
                      name  varchar(32),
                      land  db2gse.ST_Polygon);
```

以下の ODBC プログラムでは、ST_InteriorRingN 関数を使って、それぞれの島のポリゴンから内部リング (湖) を折れ線として抽出します。 length 関数によって戻される折れ線の周囲の長さを合計して、その島の ID とともに表示します。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "sg.h"
#include "sgerr.h"
#include "sqlcli1.h"
/***                          ***
 *** Change these constants ***
 ***                          ***/
#define USER_NAME   "sdetest"   /* your user name */
#define USER_PASS   "acid.rain" /* your user password */
#define DB_NAME     "mydb"      /* database to connect to */
static void check_sql_err (SQLHDBC  handle,
                           SQLHSTMT hstmt,
                           LONG          rc,
                           CHAR     *str);
void main( argc, argv )
int argc;
char *argv[];
{
  SQLHDBC       handle;
  SQLHENV       henv;
  CHAR          sql_stmt[256];
```

```
  LONG          rc,
                total_perimeter,
                num_lakes,
                lake_number,
                island_id,
                lake_perimeter;
  SQLHSTMT      island_cursor,
                lake_cursor;
  SDWORD        pcbvalue,
                id_ind,
                lake_ind,
                length_ind;
/* Allocate memory for the ODBC environment handle henv and initialize the application. */
  rc = SQLAllocEnv (&henv);
    if (rc != SQL_SUCCESS)
  {
    printf ("SQLAllocEnv failed with %d¥n", rc);
    exit(0);
  }
/* Allocate memory for a connection handle within the henv environment. */
  rc = SQLAllocConnect (henv, &handle);
    if (rc != SQL_SUCCESS)
  {
    printf ("SQLAllocConnect failed with %d¥n", rc);
    exit(0);
  }
/* Load the ODBC driver and connect to the data source identified by the database,
   user, and password.*/
  rc = SQLConnect (handle,
                   (UCHAR *)DB_NAME,
                   SQL_NTS,
                   (UCHAR *)USER_NAME,
                   SQL_NTS,
                   (UCHAR *)USER_PASS,
                   SQL_NTS);
  check_sql_err (handle, NULL, rc, "SQLConnect");
/* Allocate memory to the SQL statement handle island_cursor. */
  rc = SQLAllocStmt (handle, &island_cursor);
  check_sql_err (handle, NULL, rc, "SQLAllocStmt");
/* Prepare and execute the query to get the island IDs and number of
   lakes (interior rings) */
  strcpy (sql_stmt, "select id, db2gse.ST_NumInteriorRings(land) from ISLANDS");
  rc = SQLExecDirect (island_cursor, (UCHAR *)sql_stmt, SQL_NTS);
  check_sql_err (NULL, island_cursor, rc, "SQLExecDirect");
/* Bind the island table's ID column to the variable island_id */
  rc = SQLBindCol (island_cursor, 1, SQL_C_SLONG, &island_id, 0, &id_ind);
  check_sql_err (NULL, island_cursor, rc, "SQLBindCol");
/* Bind the result of numinteriorrings(land) to the num_lakes variable. */
  rc = SQLBindCol (island_cursor, 2, SQL_C_SLONG, &num_lakes, 0, &lake_ind);
  check_sql_err (NULL, island_cursor, rc, "SQLBindCol");
/* Allocate memory to the SQL statement handle lake_cursor. */
rc = SQLAllocStmt (handle, &lake_cursor);
  check_sql_err (handle, NULL, rc, "SQLAllocStmt");
/* Prepare the query to get the length of an interior ring. */
  strcpy (sql_stmt,
          "select Length(db2gse.ST_InteriorRingN(land, cast (? as
           integer))) from ISLANDS where id = ?");
  rc = SQLPrepare (lake_cursor, (UCHAR *)sql_stmt, SQL_NTS);
  check_sql_err (NULL, lake_cursor, rc, "SQLPrepare");
/* Bind the lake_number variable as the first input parameter */
  pcbvalue = 0;
  rc = SQLBindParameter (lake_cursor, 1, SQL_PARAM_INPUT, SQL_C_LONG,
       SQL_INTEGER, 0, 0, &lake_number, 0, &pcbvalue);
  check_sql_err (NULL, lake_cursor, rc, "SQLBindParameter");
/* Bind the island_id as the second input parameter */
  pcbvalue = 0;
  rc = SQLBindParameter (lake_cursor, 2, SQL_PARAM_INPUT, SQL_C_LONG,
       SQL_INTEGER, 0, 0, &island_id, 0, &pcbvalue);
  check_sql_err (NULL, lake_cursor, rc, "SQLBindParameter");
/* Bind the result of the Length(db2gse.ST_InteriorRingN(land, cast (? as integer)))
   to the variable lake_perimeter */
  rc = SQLBindCol (lake_cursor, 1, SQL_C_SLONG, &lake_perimeter, 0,
```

```
                    &length_ind);
  check_sql_err (NULL, island_cursor, rc, "SQLBindCol");
  /* Outer loop, get the island ids and the number of lakes (interior rings) */
  while (SQL_SUCCESS == rc)
  {
    /* Fetch an island */
    rc = SQLFetch (island_cursor);
    if (rc != SQL_NO_DATA)
    {
      check_sql_err (NULL, island_cursor, rc, "SQLFetch");
      /* Inner loop, for this island, get the perimeter of all of
         its lakes (interior rings) */
      for (total_perimeter = 0,lake_number = 1;
           lake_number <= num_lakes;
           lake_number++)
      {
        rc = SQLExecute (lake_cursor);
        check_sql_err (NULL, lake_cursor, rc, "SQLExecute");
        rc = SQLFetch (lake_cursor);
        check_sql_err (NULL, lake_cursor, rc, "SQLFetch");
        total_perimeter += lake_perimeter;
        SQLFreeStmt (lake_cursor, SQL_CLOSE);
      }
    }
/* Display the Island id and the total perimeter of its lakes. */
    printf ("Island ID = %d,  Total lake perimeter = %d¥n",
             island_id,total_perimeter);
  }
  SQLFreeStmt (lake_cursor, SQL_DROP);
  SQLFreeStmt (island_cursor, SQL_DROP);
        SQLDisconnect (handle);
        SQLFreeConnect (handle);
  SQLFreeEnv (henv);
  printf( "¥nTest Complete ...¥n" );
}
static void check_sql_err (SQLHDBC handle, SQLHSTMT  hstmt, LONG rc,
                           CHAR *str)
{
    SDWORD dbms_err = 0;
    SWORD  length;
    UCHAR   err_msg[SQL_MAX_MESSAGE_LENGTH], state[6];
    if (rc != SQL_SUCCESS)
    {
      SQLError (SQL_NULL_HENV, handle, hstmt, state, &dbms_err,
                err_msg, SQL_MAX_MESSAGE_LENGTH - 1, &length);
      printf ("%s ERROR (%d): DBMS code:%d, SQL state: %s, message: ¥n %s¥n",
              str, rc, dbms_err, state, err_msg);
      if (handle)
      {
        SQLDisconnect (handle);
        SQLFreeConnect (handle);
      }
      exit(1);
    }
}
```

ST_Intersection

ST_Intersection は 2 つの図形オブジェクトを引き数とし、それらの交差セットを図形オブジェクトとして戻します。

構文

db2gse.ST_Intersection(g1 db2gse.ST_Geometry, g2 db2gse.ST_Geometry)

戻りタイプ

db2gse.ST_Geometry

例

消防本部長は、危険な廃棄物汚染の可能性のある範囲と交差している病院、学校、および養護施設の地域を把握する必要があります。

重要地域は、以下の CREATE TABLE ステートメントを用いて作成される SENSITIVE_AREAS 表に格納されます。 ZONE 列は、それぞれの重要地域の輪郭を格納するポリゴンとして定義されます。

```
CREATE TABLE SENSITIVE_AREAS (id    integer,
                              name  varchar(128),
                              size  float,
                              type  varchar(10),
                              zone  db2gse.ST_Polygon);
```

危険地帯は、以下の CREATE TABLE ステートメントを用いて作成される HAZARDOUS_SITES 表に格納されます。ポイントとして定義された LOCATION 列には、それぞれの危険地帯の地理上の中心の位置が格納されます。

```
CREATE TABLE HAZARDOUS_SITES (site_id   integer,
                              name      varchar(128),
                              location  db2gse.ST_Point);
```

buffer 関数は、危険な廃棄物施設の場所を取り囲む 5 マイルの緩衝地帯を生成します。 ST_Intersection 関数は、危険な廃棄物施設の緩衝地帯のポリゴンと重要地域との交差部分からポリゴンを生成します。 ST_Area 関数は、交差ポリゴンの面積を戻します。この面積は、SUM 関数によってそれぞれの危険地帯ごとに合計されます。 GROUP BY 文節により、照会は、交差している部分の面積を危険廃棄物地帯の site_ID ごとに集計します。

```
SELECT hs.name,SUM(db2gse.ST_Area(db2gse.ST_Intersection (sa.zone,
db2gse.ST_buffer hs.location,(5 * 5280))))
FROM SENSITIVE_AREAS sa, HAZARDOUS_SITES hs
GROUP BY hs.site_id;
```

図35 で、円は危険廃棄物地帯を取り囲む緩衝地帯のポリゴンを表しています。これらの緩衝地帯ポリゴンと重要地域ポリゴンとの交差によって、別の 3 つのポリゴンが作成されます。左上隅の病院は 2 回交差していますが、右下隅の学校は 1 回だけ交差しています。

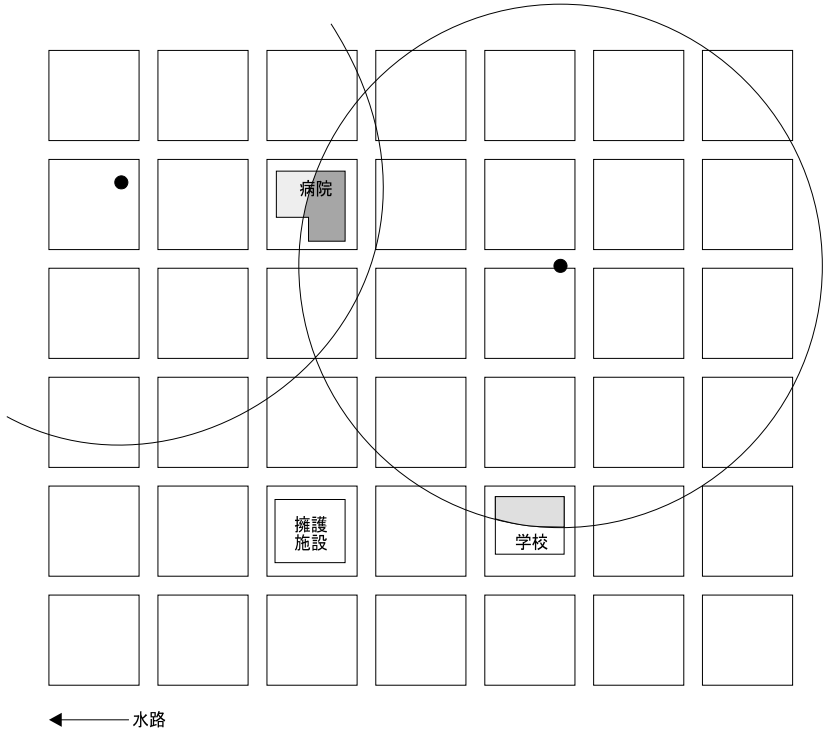


図 35. ST_Intersection を用いて、それぞれの建物の中でどの程度の広さの地域が危険廃棄物の影響を受ける可能性があるかを判別する

ST_Intersects

ST_Intersects は 2 つの図形を引き数とし、 2 つの図形の交差が空のセットを生成しない場合に 1 (TRUE) を戻します。そうでない場合、0 (FALSE) を戻します。

構文

db2gse.ST_Intersects(g1 db2gse.ST_Geometry, g2 db2gse.ST_Geometry)

戻りタイプ

整数

例

消防本部長は、危険廃棄物施設から半径 5 マイル以内にあるすべての重要地域のリストを必要としています。

重要地域は、以下の CREATE TABLE ステートメントを用いて作成される SENSITIVE_AREAS 表に格納されます。 ZONE 列は、それぞれの重要地域の輪郭を格納するポリゴンとして定義されます。

```
CREATE TABLE SENSITIVE_AREAS (id        integer,
                              name      varchar(128),
                              size      float,
                              type      varchar(10),
                              zone      db2gse.ST_Polygon);
```

危険地帯は、以下の CREATE TABLE ステートメントを用いて作成される HAZARDOUS_SITES 表に格納されます。ポイントとして定義された LOCATION 列には、それぞれの危険地帯の地理上の中心の位置が格納されます。

```
CREATE TABLE HAZARDOUS_SITES (site_id   integer,
                              name      varchar(128),
                              location  db2gse.ST_Point);
```

この照会によって、危険地帯から 5 マイルの緩衝地帯と交差する重要地域の名前と、その重要地域に対応する危険地帯の名前がリストされます。

```
SELECT sa.name, hs.name
FROM SENSITIVE_AREAS sa, HAZARDOUS_SITES hs
WHERE db2gse.ST_Intersects(db2gse.ST_Buffer(hs.location,(5 * 5280)),sa.zone) = 1;
```

ST_IsClosed

ST_IsClosed は折れ線または複数折れ線を引き数とし、そのストリングが閉じていれば 1 (TRUE) を戻します。そうでなければ 0 (FALSE) を戻します。

構文

db2gse.ST_IsClosed(c db2gse.ST_Curve)
db2gse.ST_IsClosed(mc db2gse.ST_MultiCurve)

戻りタイプ

整数

例

以下の CREATE TABLE ステートメントによって CLOSED_LINESTRING 表が作成されます。この表には 1 つの折れ線列があります。

```
CREATE TABLE CLOSED_LINESTRING (ln1 db2gse.ST_LineString)
```

以下の INSERT ステートメントによって 2 つのレコードが CLOSED_LINESTRING 表に挿入されます。最初のレコードは閉じた折れ線ではありませんが、2 番目は閉じた折れ線です。

```
INSERT INTO CLOSED_LINESTRING
VALUES(db2gse.ST_LineFromText('linestring  (10.02 20.01,10.32 23.98,11.92 25.64)',
                              db2gse.coordref()..srid(0)))
INSERT INTO CLOSED_LINESTRING
VALUES(db2gse.ST_LineFromText('linestring (10.02 20.01,11.92 35.64,25.02 34.15,
                                   19.15 33.94,10.02 20.01)',
                              db2gse.coordref()..srid(0)))
```

以下の SELECT ステートメントおよび対応する結果セットによって、ST_IsClosed 関数の出力が表示されます。最初の行は折れ線が閉じていないので 0 を戻しますが、 2 番目の行は折れ線が閉じているので 1 を戻します。

```
SELECT db2gse.ST_IsClosed(ln1) "Is it closed" FROM CLOSED_LINESTRING
Is it closed
------------
0
1
  2 record(s) selected.
```

以下の CREATE TABLE ステートメントによって CLOSED_MULTILINESTRING 表が作成されます。この表には 1 つの複数折れ線列があります。

```
CREATE TABLE CLOSED_MULTILINESTRING (mln1 db2gse.ST_MultiLineString)
```

以下の INSERT ステートメントによって、閉じていない複数折れ線のレコードと、閉じた複数折れ線のレコードが CLOSED_MULTILINESTRING に挿入されます。

```
INSERT INTO CLOSED_MULTILINESTRING
VALUES(db2gse.ST_MLineFromText('multilinestring ((10.02 20.01,10.32 23.98,11.92 25.64),
                                        (9.55 23.75,15.36 30.11))',
                               db2gse.coordref()..srid(0)))
INSERT INTO CLOSED_MULTILINESTRING
VALUES(db2gse.ST_MLineFromText('multilinestring ((10.02 20.01,11.92 35.64,25.02 34.15,
                                         19.15 33.94,10.02 20.01),
                                        (51.71 21.73,73.36 27.04,71.52 32.87,
                                         52.43 31.90,51.71 21.73))',
                               db2gse.coordref()..srid(0)))
```

以下の SELECT ステートメントおよび対応する結果セットによって、ST_IsClosed 関数の出力が表示されます。最初の行は複数折れ線が閉じていないので 0 を戻しますが、 2 番目の行は複数折れ線が閉じているので 1 を戻します。折れ線要素がすべて閉じている場合、その複数折れ線は閉じています。

```
SELECT db2gse.ST_IsClosed(mln1) "Is it closed" FROM CLOSED_MULTILINESTRING
Is it closed
------------
           0
           1
  2 record(s) selected.
```

ST_IsEmpty

ST_IsEmpty は図形オブジェクトを引き数とし、それが空であれば 1 (TRUE) を戻します。そうでなければ 0 (FALSE) を戻します。

構文

db2gse.ST_IsEmpty(g db2gse.ST_Geometry)

戻りタイプ

整数

例

以下のような CREATE TABLE ステートメントによって、 2 つの列のある EMPTY_TEST 表が作成されます。 GEOTYPE には、G1 図形列に格納されているサブクラスのデータ・タイプが格納されます。

```
CREATE TABLE EMPTY_TEST (geotype varchar(20), g1 db2gse.ST_Geometry)
```

以下の INSERT ステートメントによって、図形サブクラスのポイント、折れ線、およびポリゴンのそれぞれについて、2 つのレコードが挿入されます。前者は空ですが、後者は空ではありません。

```
INSERT INTO EMPTY_TEST
VALUES('Point', db2gse.ST_PointFromText('point (10.02 20.01)',
db2gse.coordref()..srid(0)))
INSERT INTO EMPTY_TEST
VALUES('Point', db2gse.ST_PointFromText('point empty', db2gse.coordref()..srid(0)))
INSERT INTO EMPTY_TEST
VALUES('Linestring', db2gse.ST_LineFromText('linestring (10.02 20.01,10.32 23.98,
                                              11.92 25.64)',
                            db2gse.coordref()..srid(0)))
INSERT INTO EMPTY_TEST
VALUES('Linestring', db2gse.ST_LineFromText('linestring  empty',
db2gse.coordref()..srid(0)))
INSERT INTO EMPTY_TEST
VALUES('Polygon', db2gse.ST_PolyFromText('polygon ((10.02 20.01,11.92 35.64,
25.02 34.15,19.15 33.94,10.02 20.01))',
                            db2gse.coordref()..srid(0)))
INSERT INTO EMPTY_TEST
VALUES('Polygon', db2gse.ST_PolyFromText('polygon  empty', db2gse.coordref()..srid(0)))
```

以下の SELECT ステートメントおよび対応する結果セットによって、GEOTYPE 列にある図形タイプと ST_IsEmpty 関数の結果が表示されます。

```
SELECT geotype, db2gse.ST_IsEmpty(g1) "It is empty" FROM EMPTY_TEST
GEOTYPE                  It is empty
-------------------      -----------
ST_Point                     0
ST_Point                   1
ST_Linestring                0
```

```
ST_Linestring                1
ST_Polygon                   0
ST_Polygon                 1
6 record(s) selected.
```

ST_IsRing

ST_IsRing は linestring を引き数とし、それがリングであれば (つまり閉じた単純な折れ線であれば)、1 (TRUE) を戻します。そうでなければ 0 (FALSE) を戻します。

構文

db2gse.ST_IsRing(c db2gse.ST_Curve)

戻りタイプ

整数

例

以下の CREATE TABLE ステートメントによって RING_LINESTRING 表が作成されます。この表には LN1 という 1 つの折れ線列があります。

```
CREATE TABLE RING_LINESTRING (ln1 db2gse.ST_LineString)
```

以下の INSERT ステートメントによって、3 つの折れ線が LN1 列に挿入されます。最初の行には、閉じていない、つまりリングでない折れ線が含まれています。 2 番目の行には、閉じた単純な折れ線、つまりリングである折れ線が含まれています。 3 番目の行には、閉じているものの単純ではない (自身の内部と交差している)、つまりリングではない折れ線が含まれています。

```
INSERT INTO RING_LINESTRING
VALUES(db2gse.ST_LineFromText('linestring  (10.02 20.01,10.32 23.98,11.92 25.64)',
                              db2gse.coordref()..srid(0)))
INSERT INTO RING_LINESTRING
VALUES(db2gse.ST_LineFromText('linestring (10.02 20.01,11.92 35.64,25.02 34.15,
                                 19.15 33.94, 10.02 20.01)',
                              db2gse.coordref()..srid(0)))
INSERT INTO RING_LINESTRING
VALUES(db2gse.ST_LineFromText('linestring (15.47 30.12,20.73 22.12,10.83 14.13,
                                 16.45 17.24,21.56 13.37,11.23 22.56,
                                 19.11 26.78,15.47 30.12)',
                              db2gse.coordref()..srid(0)))
```

以下の SELECT ステートメントおよび対応する結果セットによって、ST_IsRing 関数の出力が表示されます。最初の行と 3 番目の行は、0 を戻します。その理由は折れ線がリングではないからですが、 2 番目の行は折れ線がリングなので 1 を戻します。

```
SELECT db2gse.ST_IsRing(ln1) "Is it ring" FROM RING_LINESTRING
Is it ring
-----------
          0
          1
          0
  3 record(s) selected.
```

ST_IsSimple

ST_IsSimple は図形オブジェクトを引き数とし、そのオブジェクトが単純であれば 1 (TRUE) を戻します。そうでなければ 0 (FALSE) を戻します。

構文

db2gse.ST_IsSimple(g db2gse.ST_Geometry)

戻りタイプ

整数

例

以下の CREATE TABLE ステートメントによって ISSIMPLE_TEST 表が作成されます。この表には 2 つの列があります。 PID 列は短整数であり、それぞれの行ごとの固有の識別子を含んでいます。 G1 図形列には、単純な図形サンプルと単純でない図形サンプルが格納されます。

```
CREATE TABLE ISSIMPLE_TEST (pid smallint, g1 db2gse.ST_Geometry)
```

以下の INSERT ステートメントによって 2 つのレコードが ISSIMPLE_TEST 表に挿入されます。最初のレコードは、自身の内部と交差しない折れ線なので、単純です。 2 番目は自身の内部と交差するので、単純ではありません。

```
INSERT INTO ISSIMPLE_TEST
VALUES (1, db2gse.ST_LineFromText('linestring (10 10, 20 20, 30 30)',
db2gse.coordref()..srid(0)))
INSERT INTO ISSIMPLE_TEST
VALUES (2, db2gse.ST_LineFromText('linestring (10 10,20 20,20 30,10 30,10 20,20 10)',
db2gse.coordref()..srid(0)))
```

以下の SELECT ステートメントおよび対応する結果セットによって、ST_IsSimple 関数の出力が表示されます。最初のレコードは折れ線が単純なので 1 を戻しますが、 2 番目のレコードは折れ線が単純でないので 0 を戻します。

```
SELECT ST_IsSimple(g1)
FROM ISSIMPLE_TEST
g1
--------------
1
0
```

ST_IsValid

ST_IsValid は ST_Geometry を引き数とし、それが有効であれば 1 (TRUE) を戻します。そうでなければ 0 (FALSE) を戻します。 DB2 データベースに挿入されている図形は、必ず DB2 地理情報エクステンダーによって地理情報データの妥当性を検査されてから挿入されるので、常に有効です。しかし、他社の DBMS の場合は入力の妥当性検査が行われない可能性もあるので、その場合は代わりにアプリケーションで妥当性検査を行う必要があります。

構文

db2gse.ST_IsValid(g db2gse.ST_Geometry)

戻りタイプ

整数

例

geotype 列および g1 列がある valid_test 表が作成されます。 geotype 列には、g1 図形列に格納されている図形サブクラスの名前が格納されます。

```
CREATE TABLE valid_test (geotype varchar(20), g1 db2gse.ST_Geometry)
```

INSERT ステートメントによって、サンプルのサブクラスが valid_test 表に挿入されます。

```
INSERT INTO valid_test VALUES(
    'Point', db2gse.ST_PointFromText('point (10.02 20.01)', db2gse.coordref()..srid(0))
)
INSERT INTO valid_test VALUES(
'Linestring',
db2gse.ST_LineFromText('linestring  (10.02 20.01,10.32 23.98,11.92 25.64)',
db2gse.coordref()..srid(0))
)
INSERT INTO valid_test VALUES(
'Polygon', db2gse.ST_PolyFromText('polygon ((10.02 20.01,11.92 35.64,25.02 34.15,
19.15 33.94,10.02 20.01))', db2gse.coordref()..srid(0))
)
INSERT INTO valid_test VALUES(
'Multipoint', db2gse.ST_MPointFromText('multipoint (10.02 20.01,10.32 23.98,
11.92 25.64)', db2gse.coordref()..srid(0))
)
INSERT INTO valid_test VALUES(
'Multilinestring', db2gse.ST_MLineFromText('multilinestring ((10.02 20.01,
10.32 23.98,11.92 25.64),(9.55 23.75,15.36 30.11))', db2gse.coordref()..srid(0))
)
INSERT INTO valid_test VALUES(
'Multipolygon',
db2gse.ST_MPolyFromText('multipolygon (((10.02 20.01,11.92 35.64,25.02 34.15,
19.15 33.94,10.02 20.01)),((51.71 21.73,73.36 27.04,71.52 32.87,
52.43 31.90,51.71 21.73)))', db2gse.coordref()..srid(0))
)
```

SELECT ステートメントによって、 geotype の次元とともに geotype 列に格納されているサブクラス名がリストされます。

```
SELECT geotype, db2gse.ST_IsValid(g1) Valid FROM valid_test

GEOTYPE              Valid
-------------------- -------------
ST_Point                     1
ST_Linestring                1
ST_Polygon                   1
ST_Multipoint                1
ST_Multilinestring           1
ST_Multipolygon              1
6 record(s) selected.
```

ST_Length

ST_Length は折れ線または複数折れ線を引き数とし、その長さを戻します。

構文

db2gse.ST_Length(c db2gse.ST_Curve)
db2gse.ST_Length(mc db2gse.ST_MultiCurve)

戻りタイプ

倍精度

例

地元の生態学者が、自治体内の水路におけるサケの群れの移動パターンを研究しています。この学者は、自治体内を流れるすべての河川とその支流の長さを把握したいと考えています。

以下の CREATE TABLE ステートメントによって WATERWAYS 表が作成されます。 ID 列と NAME 列は、表に格納されているそれぞれの河川とその支流を識別します。河川とその支流は複数の折れ線の集合体となっていることが多いので、WATER 列は複数折れ線です。

```
CREATE TABLE WATERWAYS (id integer, name varchar(128),
water      db2gse.ST_MultiLineString);
```

以下の SELECT ステートメントでは、ST_Length 関数を使用して、自治体の内側を流れる個々の水路の名前と長さが戻されます。

```
SELECT name, db2gse.ST_Length(water) "Length"
FROM WATERWAYS;
```

241ページの図36 では、自治体の境界の内側を流れるある河川系を示しています。

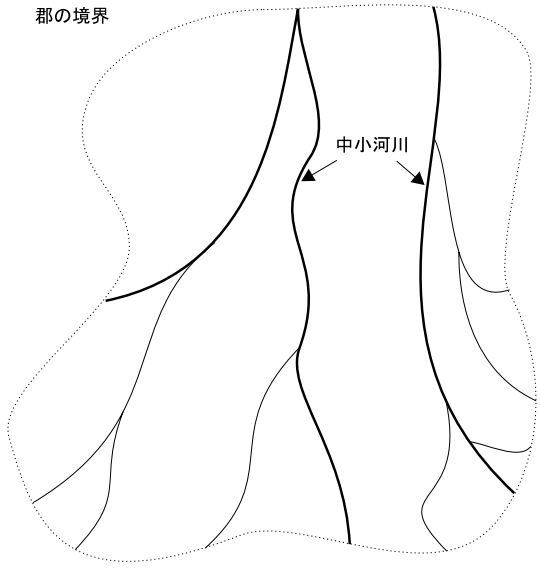


図 36. ST_Length を用いて自治体内の多数の水路の全長を判別する

ST_LineFromText

ST_LineFromText は折れ線型の事前割り当てのテキスト表現と地理情報参照システム ID を引き数とし、折れ線を戻します。

構文

db2gse.ST_LineFromText(lineStringTaggedText Varchar(4000), cr db2gse.coordref)

戻りタイプ

db2gse.ST_LineString

例

以下の CREATE TABLE ステートメントによって LINESTRING_TEST 表が作成されます。この表には 1 つの LN1 折れ線列があります。

```
CREATE TABLE LINESTRING_TEST (ln1 db2gse.ST_LineString)
```

以下の INSERT ステートメントは、ST_LineFromText 関数を使って折れ線を LN1 列に挿入します。

```
INSERT INTO LINESTRING_TEST
VALUES (db2gse.ST_LineFromText('linestring(10.01 20.03,20.94 21.34,35.93 19.04)',
db2gse.coordref()..srid(0)))
```

ST_LineFromWKB

ST_LineFromWKB は折れ線型の事前割り当てのバイナリー表現と地理情報参照システム ID を引き数とし、折れ線を戻します。

構文

db2gse.ST_LineFromWKB(WKBLineString Blob(1M), cr db2gse.coordref)

戻りタイプ

db2gse.ST_LineString

例

以下のコード断片では、SEWERLINES 表に、それぞれの下水路の固有の ID、サイズ・クラス、および図形を入れます。

3 つの列のある SEWERLINES 表が作成されます。最初の列である SEWER_ID は、それぞれの下水路を一意的に識別します。 2 番目の列である CLASS は整数タイプで、下水路のタイプを識別します。通常は下水路の容量と関連付けられています。 3 番目の列である SEWER は折れ線タイプで、下水路の図形を格納します。

```
CREATE TABLE SEWERLINES (sewer_id   integer,
                         class  integer,
                         sewer  db2gse.ST_LineString);
/* Create the SQL insert statement to populate the sewer_id, size class
   and the sewer linestring. The question marks are parameter markers that
   indicate the sewer_id, class and sewer geometry values that will be
   retrieved at runtime. */
strcpy (wkb_sql,"insert into sewerlines (sewer_id,class,sewer)
values (?,?, db2gse.ST_LineFromWKB (cast(? as blob(1m)), db2gse.coordref()..srid(0)))");
/* Allocate memory for the SQL statement handle and associate the
   statement handle with the connection handle. */
rc = SQLAllocStmt (handle, &hstmt);
/* Prepare the SQL statement for execution. */
rc = SQLPrepare (hstmt, (unsigned char *)wkb_sql, SQL_NTS);
/* Bind the integer sewer_id value to the first parameter. */
pcbvalue1 = 0;
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_SLONG,
   SQL_INTEGER, 0, 0, &sewer_id, 0, &pcbvalue1);
/* Bind the integer class value to the second parameter. */
pcbvalue2 = 0;
rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_SLONG,
   SQL_INTEGER, 0, 0, &sewer_class, 0, &pcbvalue2);
/* Bind the shape to the third parameter. */
pcbvalue3 = blob_len;
rc = SQLBindParameter (hstmt, 3, SQL_PARAM_INPUT, SQL_C_BINARY,
     SQL_BLOB, blob_len, 0, sewer_wkb, blob_len, &pcbvalue3);
/* Execute the insert statement. */
rc = SQLExecute (hstmt);
```

ST_MLineFromText

ST_MLineFromText は、複数折れ線型の事前割り当てテキスト表現と地理情報参照システム ID を引き数とし、複数折れ線を戻します。

構文

db2gse.ST_MLineFromText(multiLineStringTaggedText String, cr db2gse.coordref)

戻りタイプ

db2gse.ST_MultiLineString

例

以下の CREATE TABLE ステートメントによって MLINESTRING_TEST 表が作成されます。 MLINESTRING_TEST には 2 つの列があります。それは、行を一意的に識別する GID 短整数列と、ML1 複数折れ線列です。

```
CREATE TABLE ST_MLINESTRING_TEST (gid smallint, ml1 db2gse.ST_MultiLineString)
```

以下の INSERT ステートメントでは、ST_MLineFromText 関数を用いて複数折れ線が挿入されます。

```
INSERT INTO MLINESTRING_TEST
VALUES (1, db2gse.ST_MLineFromText('multilinestring((10.01 20.03,10.52 40.11,30.29 41.56,
                                        31.78 10.74),
                                       (20.93 20.81, 21.52 40.10))',
                                db2gse.coordref()..srid(0)))
```

ST_MLineFromWKB

ST_MLineFromWKB は、複数折れ線型の事前割り当てバイナリー表現と地理情報参照システム ID を引き数とし、複数折れ線を戻します。

構文

db2gse.ST_MLineFromWKB(WKBMultiLineString Blob(1M), cr db2gse.coordref)

戻りタイプ

db2gse.ST_MultiLineString

例

以下のコード断片では、WATERWAYS 表に、固有の id、name、および water 複数折れ線を入れます。

WATERWAYS 表が作成されます。この表には、表に格納されているそれぞれの河川とその支流を識別する ID 列と NAME 列があります。河川とその支流は複数の折れ線の集合体となっていることが多いので、WATER 列は複数折れ線です。

```
CREATE TABLE WATERWAYS (id        integer,
                        name      varchar(128),
                        water      db2gse.ST_MultiLineString);
/* Create the SQL insert statement to populate the id, name and
   multilinestring. The question marks are parameter markers that
   indicate the id, name and water values that will be retrieved at
   runtime. */
strcpy (shp_sql,"insert into WATERWAYS (id,name,water)
values (?,?, db2gse.ST_MLineFromWKB (cast(? as blob(1m)),
db2gse.coordref()..srid(0)))");
/* Allocate memory for the SQL statement handle and associate the
   statement handle with the connection handle. */
rc = SQLAllocStmt (handle, &hstmt);
/* Prepare the SQL statement for execution. */
rc = SQLPrepare (hstmt, (unsigned char *)shp_sql, SQL_NTS);
/* Bind the integer id value to the first parameter. */
pcbvalue1 = 0;
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_SLONG,
   SQL_INTEGER, 0, 0, &id, 0, &pcbvalue1);
/* Bind the varchar name value to the second parameter. */
pcbvalue2 = name_len;
rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR,
     SQL_CHAR, name_len, 0, &name, name_len, &pcbvalue2);
/* Bind the shape to the third parameter. */
pcbvalue3 = blob_len;
rc = SQLBindParameter (hstmt, 3, SQL_PARAM_INPUT, SQL_C_BINARY,
     SQL_BLOB, blob_len, 0, water_shape, blob_len, &pcbvalue3);
/* Execute the insert statement. */
rc = SQLExecute (hstmt);
```

ST_MPointFromText

ST_MPointFromText は、複数ポイント型の事前割り当てテキスト表現と地理情報参照システム ID を引き数とし、複数ポイントを戻します。

構文

db2gse.ST_MPointFromText(multiPointTaggedText Varchar(4000), cr db2gse.coordref)

戻りタイプ

db2gse.ST_MultiPoint

例

以下の CREATE TABLE ステートメントによって MULTIPOINT_TEST 表が作成されます。この表には 1 つの複数ポイント列 MPT1 があります。

```
CREATE TABLE MULTIPOINT_TEST (mpt1 db2gse.ST_MultiPoint)
```

以下の INSERT ステートメントは、 ST_MPointFromText 関数を使って複数ポイントを MPT1 列に挿入します。

```
INSERT INTO MULTIPOINT_TEST
VALUES (1, db2gse.ST_MPointFromText('multipoint(10.01 20.03,10.52 40.11,
                                    30.29 41.56,31.78 10.74)',
                                db2gse.coordref()..srid(0)))
```

ST_MPointFromWKB

ST_MPointFromWKB は、複数ポイント型の事前割り当てバイナリー表現と地理情報参照システム ID を引き数とし、複数ポイントを戻します。

構文

db2gse.ST_MPointFromWKB(WKBMultiPoint Blob(1M), cr db2gse.coordref)

戻りタイプ

db2gse.ST_MultiPoint

例

以下のコード断片では、SPECIES_SITINGS 表にデータを入れます。

3 つの列を持つ SPECIES_SITINGS 表が作成されます。 SPECIES 列および GENUS 列はそれぞれの行を一意的に識別し、 SITINGS 複数ポイント列には種類ごとの生息地域が格納されます。

```
CREATE TABLE SPECIES_SITINGS (species  varchar(32),
                              genus varchar(32),
                              sitings  db2gse.ST_MultiPoint);
/* Create the SQL insert statement to populate the species, genus and
   sitings. The question marks are parameter markers that
   indicate the species, genus and sitings values that will be retrieved at
   runtime. */
strcpy (wkb_sql,"insert into SPECIES_SITINGS (species,genus,sitings)
values (?,?, db2gse.ST_MPointFromWKB (cast(? as blob(1m)),
db2gse.coordref()..srid(0)))");
/* Allocate memory for the SQL statement handle and associate the
   statement handle with the connection handle. */
rc = SQLAllocStmt (handle, &hstmt);
/* Prepare the SQL statement for execution. */
rc = SQLPrepare (hstmt, (unsigned char *)wkb_sql, SQL_NTS);
/* Bind the varchar species value to the first parameter. */
pcbvalue1 = species_len;
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR,
   SQL_CHAR, species_len, 0, &species, species_len, &pcbvalue1);
/* Bind the varchar genus value to the second parameter. */
pcbvalue2 = genus_len;
rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR,
   SQL_CHAR, genus_len, 0, &name, genus_len, &pcbvalue2);
/* Bind the shape to the third parameter. */
pcbvalue3 = sitings_len;
rc = SQLBindParameter (hstmt, 3, SQL_PARAM_INPUT, SQL_C_BINARY,
     SQL_BLOB, sitings_len, 0, sitings_wkb, sitings_len, &pcbvalue3);
/* Execute the insert statement. */
rc = SQLExecute (hstmt);
```

ST_MPolyFromText

ST_MPolyFromText は、複数ポリゴン型の事前割り当てテキスト表現と地理情報参照システム ID を引き数とし、複数ポリゴンを戻します。

構文

db2gse.ST_MPolyFromText(multiPolygonTaggedText Varchar(4000), cr db2gse.coordref)

戻りタイプ

db2gse.ST_MultiPolygon

例

以下の CREATE TABLE ステートメントによって MULTIPOLYGON_TEST 表が作成されます。この表には 1 つの複数折れ線列 MPL1 があります。

```
CREATE TABLE MULTIPOLYGON_TEST (mpl1 db2gse.ST_MultiPolygon)
```

以下の INSERT ステートメントでは、 ST_MPolyFromText 関数を用いて複数ポリゴンを MPL1 列に挿入します。

```
INSERT INTO MULTIPOLYGON_TEST VALUES (
db2gse.ST_MPolyFromText('multipolygon(((10.01 20.03,10.52 40.11,30.29 41.56,31.78
10.74,10.01 20.03),(21.23 15.74,21.34 35.21,28.94 35.35,29.02 16.83,21.23
15.74)),((40.91 10.92,40.56 20.19,50.01 21.12,51.34 9.81,40.91 10.92)))',
                                db2gse.coordref()..srid(0)))
```

ST_MPolyFromWKB

ST_MPolyFromWKB は、複数ポリゴン型の事前割り当てバイナリー表現と地理情報参照システム ID を引き数とし、複数ポリゴンを戻します。

構文

db2gse.ST_MPolyFromWKB(WKBMultiPolygon Blob(1M), cr db2gse.coordref)

戻りタイプ

db2gse.ST_MultiPolygon

例

以下のコード断片では、LOTS 表にデータを入れます。

LOTS 表には、それぞれの敷地を一意的に識別する LOT_ID、および敷地境界線の図形を含む LOT 複数ポリゴンが格納されています。

```
CREATE TABLE LOTS (  lot_id integer, lot db2gse.ST_MultiPolygon );
/* Create the SQL insert statement to populate the lot_id, and lot. The
   question marks are parameter markers that indicate the lot_id, and lot
   values that will be retrieved at runtime. */
strcpy (wkb_sql,"insert into LOTS (lot_id,lot)
values (?, db2gse.ST_MPolyFromWKB (cast(? as blob(1m)),
db2gse.coordref()..srid(0)))");
/* Allocate memory for the SQL statement handle and associate the
   statement handle with the connection handle. */
rc = SQLAllocStmt (handle, &hstmt);
/* Prepare the SQL statement for execution. */
rc = SQLPrepare (hstmt, (unsigned char *)wkb_sql, SQL_NTS);
/* Bind the lot_id integer value to the first parameter. */
pcbvalue1 = 0;
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_INTEGER,
     SQL_INTEGER, 0, 0, &lot_id, 0, &pcbvalue1);
/* Bind the lot shape to the second parameter. */
pcbvalue2 = lot_len;
rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_BINARY,
     SQL_BLOB, lot_len, 0, lot_wkb, lot_len, &pcbvalue2);
/* Execute the insert statement. */
rc = SQLExecute (hstmt);
```

ST_NumGeometries

ST_NumGeometries はコレクションを引き数とし、コレクション内の図形の数を戻します。

構文

db2gse.ST_NumGeometries(g db2gse.ST_GeomCollection)

戻りタイプ

整数

例

都市開発担当者は、それぞれの建物のフットプリントに関連付けられた、個別の建物の実際の数を知っていなければなりません。

建物のフットプリントは、以下のような CREATE TABLE ステートメントを用いて作成された BUILDINGFOOTPRINTS 表の中に収められています。

```
CREATE TABLE BUILDINGFOOTPRINTS (   building_id integer,
                                    lot_id      integer,
                                    footprint   db2gse.ST_MultiPolygon);
```

以下の SELECT ステートメントでは、ST_NumGeometries 関数を使って、それぞれの建物を一意的に識別する BUILDING_ID と各フットプリント内の建物の数をリストします。

```
SELECT building_id, db2gse.ST_NumGeometries (footprint) "Number of buildings"
FROM BUILDINGFOOTPRINTS;
```

ST_NumInteriorRing

ST_NumInteriorRing はポリゴンを引き数とし、その内部リングの数を戻します。

構文

db2gse.NumInteriorRing(p db2gse.ST_Polygon)

戻りタイプ

整数

例

南洋諸島で鳥の群生の調査を計画しているある鳥類学者は、特定の採食地域が淡水湖のある島に限定されていることを知りました。そこで、この学者はどの島に 1 つまたは複数の湖があるか把握したいと考えます。

以下の CREATE TABLE ステートメントによって ISLANDS 表が作成されます。 ISLANDS 表の ID 列と NAME 列はそれぞれの島を識別し、 LAND ポリゴン列にはその島の図形が格納されています。

```
CREATE TABLE ISLANDS (id integer, name varchar(32), land db2gse.ST_Polygon);
```

内部リングは湖を表しているため、ST_NumInteriorRing 関数を用いて、少なくとも 1 つの内部リングのある島だけをリストします。

```
SELECT name FROM ISLANDS WHERE db2gse.ST_NumInteriorRing(land) > 0;
```

ST_NumPoints

ST_NumPoints は折れ線を引き数とし、そのポイントの数を戻します。

構文

db2gse.ST_NumPoints(l db2gse.ST_LineString)

戻りタイプ

整数

例

以下の CREATE TABLE ステートメントによって NUMPOINTS_TEST 表が作成されます。 GEOTYPE 列には、G1 図形列に格納されている図形タイプが格納されます。

```
CREATE TABLE NUMPOINTS_TEST (geotype varchar(12), g1 db2gse.ST_Geometry)
```

以下の INSERT ステートメントは、折れ線を挿入します。

```
INSERT INTO NUMPOINTS_TEST VALUES( linestring,
db2gse.ST_LineFromText('linestring (10.02 20.01, 23.73 21.92)',
db2gse.coordref()..srid(0)))
```

以下の SELECT ステートメントおよび対応する結果セットによって、図形のタイプとそれぞれの図形に含まれるポイントの数がリストされます。

```
SELECT geotype, db2gse.ST_NumPoints(g1)
FROM NUMPOINTS_TEST
GEOTYPE      Number of points
------------ ----------------
ST_linestring          2
  1 record(s) selected.
```

ST_OrderingEquals

ST_OrderingEquals は 2 つの図形を比較して、図形が等しく座標の順序が同じであれば 1 (TRUE) を戻し、そうでなければ 0 (FALSE) を戻します。

構文

db2gse.ST_OrderingEquals(g1 db2gse.ST_Geometry, g2 db2gse.ST_Geometry)

戻りタイプ

整数

例

以下の CREATE TABLE ステートメントによって LINESTRING_TEST 表が作成されます。この表には 2 つの折れ線列 L1、L2 があります。

```
CREATE TABLE LINESTRING_TEST (lid integer, l1 db2gse.ST_LineString,
l2 db2gse.ST_LineString);
```

以下の INSERT ステートメントは、等しくて座標の順序も同じである 2 つの折れ線列を L1 と L2 に挿入します。

```
INSERT INTO linestring_test VALUES (1,
db2gse.LineFromText('linestring (10.01 20.02, 21.50 12.10)', db2gse.coordref()..srid(0)),
db2gse.LineFromText('linestring (10.01 20.02, 21.50 12.10)',
db2gse.coordref()..srid(0)));
```

以下の INSERT ステートメントは、等しいが座標の順序が同じでない 2 つの折れ線列を L1 と L2 に挿入します。

```
INSERT INTO linestring_test VALUES (2,
db2gse.LineFromText('linestring (10.01 20.02, 21.50 12.10)', db2gse.coordref()..srid(0)),
db2gse.LineFromText('linestring (21.50 12.10,10.01 20.02)', db2gse.coordref()..srid(0)));
```

以下の SELECT ステートメントおよび対応する結果セットは、座標の順序に関係なく ST_Equals 関数により 1 (TRUE) が戻されることを示しています。ST_OrderingEquals 関数によって 0 (FALSE) が戻されるのは、図形が等しくなく座標の順序も同じでない場合です。

```
SELECT lid, db2gse.ST_Equals(l1,l2) equals, db2gse.ST_OrderingEquals(l1,l2)
OrderingEquals
FROM linestring_test
lid   equals   OrderingEquals
---   ------   -----------
 1       1           1
 2       1           0
```

ST_Overlaps

ST_Overlaps は 2 つの図形オブジェクトを引き数とします。それらのオブジェクトの交差によって同じ次元の図形オブジェクトが形成され、かつどちらのソース・オブジェクトとも等しくない場合に、1 (TRUE) を戻します。そうでない場合、0 (FALSE) を戻します。

構文

db2gse.ST_Overlaps(g1 db2gse.ST_Geometry, g2 db2gse.ST_Geometry)

戻りタイプ

整数

例

自治体の行政責任者が、半径 5 マイルが重要地域と重なっている危険な廃棄物地帯のリストを必要としています。

以下の CREATE TABLE ステートメントによって SENSITIVE_AREAS 表が作成されます。 SENSITIVE_AREAS 表には、汚染の恐れがある公共施設について記述する複数の列と、それらの公共施設のポリゴン図形を格納する ZONE 列が含まれています。

```
CREATE TABLE SENSITIVE_AREAS (id        integer,
                              name      varchar(128),
                              size      float,
                              type      varchar(10),
                              zone      db2gse.ST_Polygon);
```

HAZARDOUS_SITES 表には、地点を識別する SITE_ID 列と NAME 列が格納されています。それぞれの地点の実際の地理上の位置は、LOCATION ポイント列に格納されています。

```
CREATE TABLE HAZARDOUS_SITES (site_id   integer,
                              name      varchar(128),
                              location  db2gse.ST_Point);
```

以下の SELECT ステートメントで、 SENSITIVE_AREAS 表と HAZARDOUS_SITES 表は、ST_Overlaps 関数によって結合されます。この関数は、SENSITIVE_AREAS 表の行のうち、 zone ポリゴンが HAZARDOUS_SITES location ポイントから半径 5 マイルの緩衝地帯と重なっているすべての行について 1 (TRUE) を戻します。

```
SELECT hs.name
FROM HAZARDOUS_SITES hs, SENSITIVE_AREAS sa
WHERE db2gse.ST_Overlaps (buffer(hs.location,(5 * 5280)),sa.zone) = 1;
```

図37 では、病院と学校が、自治体内の 2 つの危険廃棄物施設から 5 マイルの範囲と重なっています。養護施設は重なっていません。

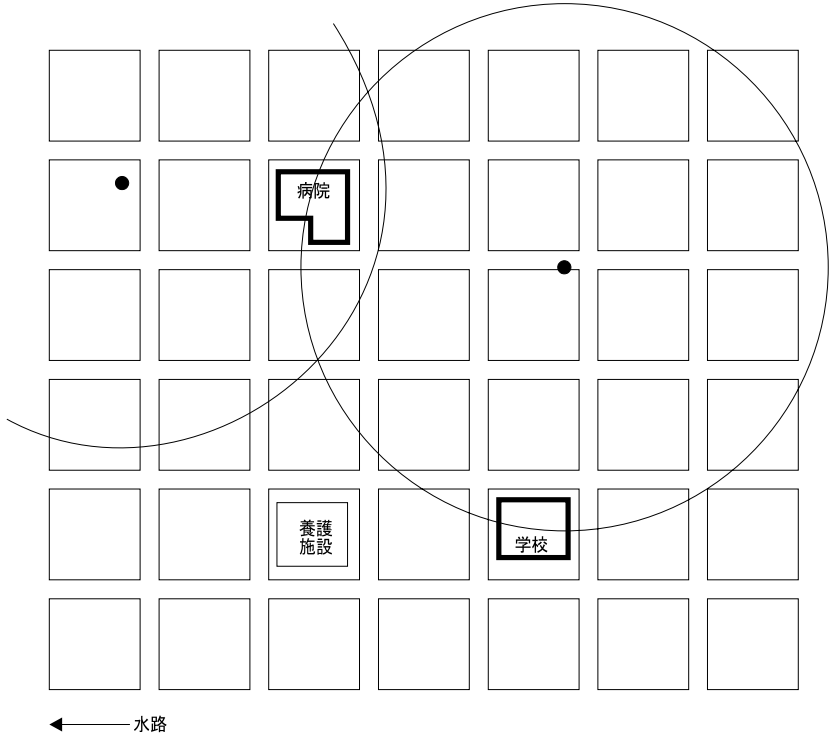


図 37. ST_Overlaps を用いて、危険廃棄物施設の地域に少なくとも一部が入っている建物を判別する

ST_Perimeter

ST_Perimeter は、ST_Surface の周囲の長さを戻します。

構文

db2gse.ST_Perimeter(s db2gse.ST_Surface)
db2gse.ST_Perimeter(ms db2gse.ST_MultiSurface)

戻りタイプ

倍精度

例

湖岸の鳥を調査している生態学者は、特定の区域内の湖の湖岸を判別する必要があります。湖は、以下のような CREATE TABLE ステートメントを用いて作成された WATERBODIES 表の中に複数ポリゴンとして収められています。

```
CREATE TABLE WATERBODIES (wbid  integer,
                          waterbody  db2gse.ST_MultiPolygon);
```

以下の SELECT ステートメントで、ST_Perimeter 関数は個々の水域の周囲の長さを戻し、 SUM 関数は周囲の長さを集計してその合計を戻します。

```
SELECT SUM(db2gse.ST_Perimeter(waterbody))
FROM waterbodies;
```

ST_PointFromText

ST_PointFromText は、ポイント型の事前割り当てテキスト表現と地理情報参照システム ID を引き数とし、ポイントを戻します。

構文

db2gse.ST_PointFromText(pointTaggedText Varchar(4000), cr db2gse.coordref)

戻りタイプ

db2gse.ST_Point

例

以下の CREATE TABLE ステートメントによって POINT_TEST 表が作成されます。この表には 1 つのポイント列 PT1 があります。

```
CREATE TABLE POINT_TEST (pt1 db2gse.ST_Point)
```

INSERT ステートメントがポイントを PT1 列に挿入する前に、ST_PointFromText 関数は、ポイント・テキスト座標をポイント形式に変換します。

```
INSERT INTO POINT_TEST VALUES (
       db2gse.ST_PointFromText ('point(10.01 20.03)', db2gse.coordref()..srid(0)))
```

ST_PointFromWKB

ST_PointFromWKB は、ポイント型の事前割り当てバイナリー表現と地理情報参照システム ID を引き数とし、ポイントを戻します。

構文

db2gse.ST_PointFromWKB(WKBPoint Blob(1M), srs SRID)

戻りタイプ

db2gse.ST_Point

例

以下のコード断片では、HAZARDOUS_SITES 表にデータを入れます。

危険地帯は、以下の CREATE TABLE ステートメントを用いて作成される HAZARDOUS_SITES 表に格納されます。ポイントとして定義された LOCATION 列には、それぞれの危険地帯の地理上の中心の位置が格納されます。

```
CREATE TABLE HAZARDOUS_SITES (site_id   integer,
                              name      varchar(128),
                              location  db2gse.ST_Point);
/* Create the SQL insert statement to populate the site_id, name and
   location. The question marks are parameter markers that indicate the
   site_id, name and location values that will be retrieved at runtime. */
strcpy (wkb_sql,"insert into HAZARDOUS_SITES (site_id, name, location)
values (?,?, db2gse.ST_PointFromWKB(cast(? as blob(1m)),
db2gse.coordref()..srid(0)))");
/* Allocate memory for the SQL statement handle and associate the
   statement handle with the connection handle. */
rc = SQLAllocStmt (handle, &hstmt);
/* Prepare the SQL statement for execution. */
rc = SQLPrepare (hstmt, (unsigned char *)wkb_sql, SQL_NTS);
/* Bind the site_id integer value to the first parameter. */
pcbvalue1 = 0;
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_INTEGER,
     SQL_INTEGER, 0, 0, &site_id, 0, &pcbvalue1);
/* Bind the name varchar value to the second parameter. */
pcbvalue2 = name_len;
rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR,
   SQL_CHAR, 0, 0, name, 0, &pcbvalue2);
/* Bind the location shape to the third parameter. */
pcbvalue3 = location_len;
rc = SQLBindParameter (hstmt, 3, SQL_PARAM_INPUT, SQL_C_BINARY,
     SQL_BLOB, location_len, 0, location_wkb, location_len, &pcbvalue3);
/* Execute the insert statement. */
rc = SQLExecute (hstmt);
```

ST_Point

ST_Point は ST_Point を戻して、X 座標、Y 座標、および地理情報参照を示します。

構文

db2gse.ST_Point(X Double, Y Double, srs SRID)

戻りタイプ

db2gse.ST_Point

例

以下の CREATE TABLE ステートメントによって POINT_TEST 表が作成されます。この表には 1 つのポイント列 PT1 があります。

```
CREATE TABLE POINT_TEST (pt1 db2gse.ST_Point)
```

ST_Point 関数は、INSERT ステートメントがポイントを PT1 列に挿入する前に、ポイント座標をポイント図形に変換します。

```
INSERT INTO point_test VALUES(
    db2gse.ST_Point(10.01,20.03, db2gse.coordref()..srid(0))
)
```

ST_PointN

ST_PointN は linestring と整数インデックスを引き数とし、折れ線のパスにある n 番目の頂点であるポイントを戻します。

構文

db2gse.ST_PointN(l db2gse.ST_Curve, n Integer)

戻りタイプ

db2gse.ST_Point

例

以下の CREATE TABLE ステートメントによって POINTN_TEST 表が作成されます。この表には、それぞれの行を一意的に識別する GID 列と LN1 折れ線列の 2 つの列があります。

```
CREATE TABLE POINTN_TEST (gid integer, ln1 db2gse.ST_LineString)
```

以下の INSERT ステートメントによって 2 つの折れ線値が挿入されます。最初の折れ線には Z 座標や測定値がありませんが、2 番目の折れ線にはどちらもあります。

```
INSERT INTO POINTN_TEST VALUES(1,
db2gse.ST_LineFromText('linestring (10.02 20.01,23.73 21.92,30.10 40.23)',
db2gse.coordref()..srid(0)))
INSERT INTO POINTN_TEST VALUES(2,
db2gse.ST_LineFromText('linestring  zm (10.02 20.01 5.0 7.0,23.73 21.92 6.5 7.1,30.10
40.23 6.9 7.2)', db2gse.coordref()..srid(0)))
```

以下の SELECT ステートメントおよび対応する結果セットによって、 GID 列と、それぞれの折れ線の 2 番目の頂点がリストされます。最初の行の結果は Z 座標や測定値のないポイントであり、 2 番目の行の結果は Z 座標と測定値のあるポイントです。ソースの折れ線内に Z 座標や測定値が存在する場合、ST_PointN 関数はポイントとともに Z 座標または測定値を戻します。

```
SELECT gid, CAST(db2gse.ST_AsText(db2gse.ST_PointN(ln1,2)) AS varchar(60))
"The 2nd vertice"
FROM POINTN_TEST
GID         The 2nd vertice
----------- ------------------------------------------------------------
          1 POINT ( 23.73000000 21.92000000)
          2 POINT ZM ( 23.73000000 21.92000000 7.00000000 7.10000000)
  2 record(s) selected.
```

ST_PointOnSurface

ST_PointOnSurface は、ポリゴンまたは複数ポリゴンを引き数とし、 ST_Point を戻します。

構文

db2gse.ST_PointOnSurface(s db2gse.ST_Surface)
db2gse.ST_PointOnSurface(ms db2gse.ST_MultiSurface)

戻りタイプ

db2gse.ST_Point

例

都市計画の担当者は、それぞれの建物フットプリントのラベル・ポイントを作成する必要があります。

建物のフットプリントは、以下のような CREATE TABLE ステートメントを用いて作成された BUILDINGFOOTPRINTS 表の中に収められています。

```
CREATE TABLE BUILDINGFOOTPRINTS (   building_id integer,
                                    lot_id      integer,
                                    footprint   db2gse.ST_MultiPolygon);
```

ST_PointOnSurface 関数は、建物フットプリントの表面に存在することが保証されているポイントを生成します。 ST_PointOnSurface 関数が戻すポイントは、AsBinaryShape 関数によって形状に変換されます。この形状は、アプリケーションで使用できるように 1 メガバイトの文字ストリングにキャストされています。

```
SELECT CAST(db2gse.AsBinaryShape(db2gse.ST_PointOnSurface(footprint)) as blob(1m))
FROM BUILDINGFOOTPRINTS;
```

ST_PolyFromText

ST_PolyFromText はポリゴン型の事前割り当てテキスト表現と地理情報参照システム ID を引き数とし、ポリゴンを戻します。

構文

db2gse.ST_PolyFromText(polygonTaggedText Varchar(4000), cr db2gse.coordref)

戻りタイプ

db2gse.ST_Polygon

例

以下の CREATE TABLE ステートメントによって POLYGON_TEST 表が作成されます。この表には 1 つのポリゴン列があります。

```
CREATE TABLE POLYGON_TEST (pl1 db2gse.ST_Polygon)
```

以下の INSERT ステートメントは、 ST_PolyFromText 関数を使ってポリゴンをポリゴン列に挿入します。

```
INSERT INTO POLYGON_TEST VALUES (1,
db2gse.ST_PolyFromText('polygon((10.01 20.03,10.52 40.11,30.29 41.56,31.78 10.74,10.01
20.03))', db2gse.coordref()..srid(0)))
```

ST_PolyFromWKB

ST_PolyFromWKB はポリゴン型の事前割り当てバイナリー表現と地理情報参照システム ID を引き数とし、ポリゴンを戻します。

構文

db2gse.ST_PolyFromWKB(WKBPolygon Blob(1M), SRID Integer)

戻りタイプ

db2gse.ST_Polygon

例

以下のコード断片では、SENSITIVE_AREAS 表にデータを入れます。

SENSITIVE_AREAS 表には、汚染の恐れがある公共施設について記述する複数の列と、それらの公共施設のポリゴン図形を格納する zone 列が含まれています。

```
CREATE TABLE SENSITIVE_AREAS (id        integer,
                              name      varchar(128),
                              size      float,
                              type      varchar(10),
                              zone      db2gse.ST_Polygon);
/* Create the SQL insert statement to populate the id, name, size, type and
   zone. The question marks are parameter markers that indicate the id,name,
   size, type and zone values that will be retrieved at runtime. */
strcpy (shp_wkb,"insert into SENSITIVE_AREAS (id, name, size, type, zone)
values (?,?,?,?, db2gse.ST_PolyFromWKB (cast(? as blob(1m)),
db2gse.coordref()..srid(0)))");
/* Allocate memory for the SQL statement handle and associate the
   statement handle with the connection handle. */
rc = SQLAllocStmt (handle, &hstmt);
/* Prepare the SQL statement for execution. */
rc = SQLPrepare (hstmt, (unsigned char *)wkb_sql, SQL_NTS);
/* Bind the id integer value to the first parameter. */
pcbvalue1 = 0;
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_INTEGER,
   SQL_INTEGER, 0, 0, &id, 0, &pcbvalue1);
/* Bind the name varchar value to the second parameter. */
pcbvalue2 = name_len;
rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR,
   SQL_CHAR, 0, 0, name, 0, &pcbvalue2);
/* Bind the size float to the third parameter. */
pcbvalue3 = 0;
rc = SQLBindParameter (hstmt, 3, SQL_PARAM_INPUT, SQL_C_FLOAT,
     SQL_REAL, 0, 0, &size, 0, &pcbvalue3);
/* Bind the type varchar to the fourth parameter. */
pcbvalue4 = type_len;
rc = SQLBindParameter (hstmt, 3, SQL_PARAM_INPUT, SQL_C_CHAR,
     SQL_VARCHAR, type_len, 0, type, type_len, &pcbvalue4);
```

```
/* Bind the zone polygon to the fifth parameter. */
pcbvalue5 = zone_len;
rc = SQLBindParameter (hstmt, 3, SQL_PARAM_INPUT, SQL_C_BINARY,
     SQL_BLOB, zone_len, 0, zone_wkb, zone_len, &pcbvalue5);
/* Execute the insert statement. */
rc = SQLExecute (hstmt);
```

ST_Polygon

ST_Polygon は、ST_LineString および地理情報参照システム ID から ST_Polygon を生成します。

構文

db2gse.ST_Polygon(l db2gse.ST_LineString, cr db2gse.coordref)

戻りタイプ

db2gse.ST_Polygon

例

以下の CREATE TABLE ステートメントによって POLYGON_TEST 表が作成されます。この表には 1 つの列 P1 があります。

```
CREATE TABLE POLYGON_TEST (p1 db2gse.ST_polygon)
```

以下の INSERT ステートメントは、ST_Polygon 関数内で ST_LineFromText 関数を使って、リング (閉じている単純な折れ線) をポリゴンに変換し、P1 列に挿入します。

```
INSERT INTO POLYGON_TEST VALUES (
db2gse.ST_Polygon(db2gse.ST_LineFromText('linestring(10.01 20.03,20.94
21.34,35.93 10.04,10.01 20.03)', db2gse.coordref()..srid(0))),
db2gse.coordref()..srid(0)))
)
```

ST_Relate

ST_Relate は 2 つの図形を比較して、 DE-9IM パターン・マトリックス・ストリングで指定されている条件を図形が満たしていれば 1 (TRUE) を戻し、そうでなければ 0 (FALSE) を戻します。 DE-9IM パターン・マトリックスに関する情報は、147ページの『述部関数』を参照してください。

構文

db2gse.ST_Relate(g1 db2gse.ST_Geometry, g2 db2gse.ST_Geometry, patternMatrix String)

戻りタイプ

整数

例

DE-9IM パターン・マトリックスは、図形を比較する装置です。この種のマトリックスには複数のタイプがあります。たとえば、*等価* パターン・マトリックスは、 2 つの図形が等しいかどうかを知らせます。

この例で、表56 の等価パターン・マトリックスは、左から右、上から下にストリングを読みます (『T*F**FFF*』)。

表 56. 等価パターン・マトリックス

		b		
		内部	**境界**	**外部**
a	**内部**	T	*	F
	境界	*	*	F
	外部	F	F	*

次に、表 RELATE_TEST は、以下の CREATE TABLE ステートメントで作成されます。

```
CREATE TABLE RELATE_TEST (rid integer, g1 db2gse.ST_Geometry,
g2 db2gse.ST_Geometry, g3 db2gse.ST_Geometry);
```

以下の INSERT ステートメントによって、サンプルのサブクラスが RELATE_TEST 表に挿入されます。

```
INSERT INTO RELATE_TEST VALUES(
       1,
        db2gse.ST_PointFromText('point (10.02 20.01)',db2gse.coordref()..srid(0),
        db2gse.ST_PointFromText('point (10.02 20.01)',db2gse.coordref()..srid(0),
        db2gse.ST_PointFromText('point (30.01 20.01)',db2gse.coordref()..srid(0)
)
```

以下の SELECT ステートメントおよび対応する結果セットによって、 geotype の次元とともに GEOTYPE 列に格納されているサブクラス名がリストされます。

```
SELECT rid, relate(g1,g2) equals, relate(g1,g3) not_equals
FROM relate_test
RID     equals    not_equals
------- -----------   -----------------
1         1              0
  1 record(s) selected.
```

ST_SRID

ST_SRID は図形オブジェクトを引き数とし、その地理情報参照システム ID を戻します。

構文

db2gse.ST_SRID(g1 db2gse.ST_Geometry)

戻りタイプ

整数

例

DB2 地理情報エクステンダーのインストール中に SPATIAL_REFERENCES 表が作成されます。図形が作成されると、その図形の SRID が SPATIAL_REFERENCES 表に入力されます。 ST_SRID 関数はその入力の値を戻します。

たとえば、図形タイプは以下の CREATE TABLE ステートメントで使われます。

```
CREATE TABLE SRID_TEST(g1 db2gse.ST_Geometry)
```

次の INSERT ステートメントでは、座標 10.01,50.76 に位置するポイント図形が図形列 G1 に挿入されます。ポイント小数点には、ST_PointFromText 関数によって作成されたときに、 srid 値として 1 が割り当てられました。

```
INSERT INTO SRID_TEST
VALUES (db2gse.ST_PointFromText('point(10.01 50.76)', db2gse.coordref()..srid(0)))
```

以下の SELECT ステートメントおよび対応する結果セットによって図示されているように、 ST_SRID 関数は、入力されたばかりの図形の地理情報参照システム ID を戻します。

```
SELECT db2gse.ST_SRID(g1) FROM SRID_TEST
g1
--------------
1
```

ST_StartPoint

ST_StartPoint は折れ線を引き数とし、その折れ線の最初のポイントを戻します。

構文

db2gse.ST_StartPoint(c db2gse.ST_Curve)

戻りタイプ

db2gse.ST_Point

例

以下の CREATE TABLE ステートメントによって STARTPOINT_TEST 表が作成されます。 STARTPOINT_TEST には 2 つの列があります。それは、表の行を一意的に識別する GID 整数列と、LN1 折れ線列です。

```
CREATE TABLE STARTPOINT_TEST (gid integer, ln1 db2gse.ST_LineString)
```

以下の INSERT ステートメントによって、折れ線が LN1 列に挿入されます。最初の折れ線には Z 座標や測定値がありませんが、2 番目の折れ線にはどちらもあります。

```
INSERT INTO STARTPOINT_TEST VALUES(1,
db2gse.ST_LineFromText('linestring (10.02 20.01,23.73
21.92,30.10 40.23)', db2gse.coordref()..srid(0)))
INSERT INTO STARTPOINT_TEST VALUES(2,
db2gse.ST_LineFromText('linestring  zm (10.02 20.01 5.0 7.0,23.73 21.92 6.5 7.1,30.10
40.23 6.9 7.2)', db2gse.coordref()..srid(0)))
```

以下の SELECT ステートメントおよび対応する結果セットは、 ST_StartPoint 関数により個々の折れ線の最初のポイントが取り出されることを示します。ST_AsText 関数はポイントをテキスト形式に変換します。リスト内の最初のポイントには Z 座標や測定値がありませんが、 2 番目のポイントには両方ともあります。これは 2 番目のポイントのソースの折れ線にあったからです。

```
SELECT gid, CAST(db2gse.ST_AsText(db2gse.ST_StartPoint (ln1)) as varchar(60))
"Startpoint"
FROM STARTPOINT_TEST
GID         Startpoint
----------- ------------------------------------------------------------
          1 POINT ( 10.02000000 20.01000000)
          2 POINT ZM ( 10.02000000 20.01000000 5.00000000 7.00000000)
  2 record(s) selected.
```

ST_SymmetricDiff

ST_SymmetricDiff は 2 つの図形オブジェクトを引き数とし、ソース・オブジェクト間の対称差である図形オブジェクトを戻します。

構文

db2gse.ST_SymmetricDiff(g1 db2gse.ST_Geometry, g2 db2gse.ST_Geometry)

戻りタイプ

db2gse.ST_Geometry

例

自治体の行政責任者は、重要地域と、危険地帯から 5 マイルの範囲に交差していない地域の面積を判別する必要があります。

以下の CREATE TABLE ステートメントによって SENSITIVE_AREAS 表が作成されます。この表には、汚染の恐れがある公共施設について記述した複数の列があります。また SENSITIVE_AREAS 表には、それらの公共施設のポリゴン図形を格納する ZONE 列も含まれています。

```
CREATE TABLE SENSITIVE_AREAS (id        integer,
                              name      varchar(128),
                              size      float,
                              type      varchar(10),
                              zone      db2gse.ST_Polygon);
```

以下の CREATE TABLE ステートメントは HAZARDOUS_SITES 表を作成します。この表には、地点を識別する SITE_ID 列と NAME 列が格納されています。それぞれの地点の実際の地理上の位置は、LOCATION ポイント列に格納されています。

```
CREATE TABLE HAZARDOUS_SITES (site_id   integer,
                              name      varchar(128),
                              location  point);
```

ST_Buffer 関数は、危険廃棄物施設の場所を取り囲む 5 マイルの緩衝地帯を生成します。 ST_SymmetricDiff 関数は、危険廃棄物施設の緩衝地帯のポリゴンと重要地域との交差部分からポリゴンを生成します。 ST_Area 関数は、それぞれの危険施設に対応する交差ポリゴンの面積を戻します。

```
SELECT sa.name, hs.name,
       db2gse.ST_Area(db2gse.ST_SymmetricDiff (db2gse.ST_Buffer(hs.location,
(5 * 5280)),sa.zone))
FROM HAZARDOUS_SITES hs, SENSITIVE_AREAS sa
```

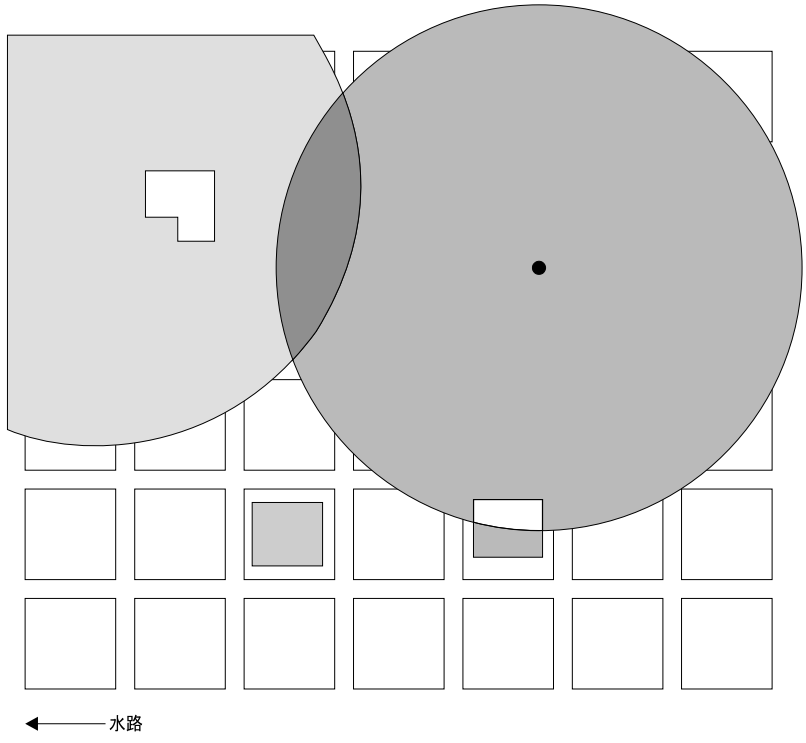


図 38. ST_SymmetricDiff を用いて、重要地域 (人が居住している建物) を含んでいない危険廃棄物の地域を判別する

図38 では、危険廃棄物施設と重要地域の対称差が、交差地域の減算結果となっています。

ST_Touches

ST_Touches は、2 つの図形に共通するポイントのうち、両方の図形の内部で交差するものが存在しない場合に、1 (TRUE) を戻します。少なくとも一方の図形が折れ線、ポリゴン、複数折れ線、または複数ポリゴンである必要があります。

構文

db2gse.ST_Touches(g1 db2gse.ST_Geometry, g2 db2gse.ST_Geometry)

戻りタイプ

整数

例

GIS 技術担当者は、下水路のうち終点で他の下水路と交差しているものをすべてリストする必要があります。

以下の CREATE TABLE ステートメントによって SEWERLINES 表が作成されます。この表には 3 つの列があります。最初の列である SEWER_ID は、それぞれの下水路を一意的に識別します。 2 番目の列である CLASS は整数タイプで、下水路のタイプを識別します。通常は下水路の容量と関連付けられています。 3 番目の列である SEWER は折れ線タイプで、下水路の図形を格納します。

```
CREATE TABLE SEWERLINES (sewer_id integer, class integer, sewer db2gse.ST_LineString);
```

以下の SELECT ステートメントでは、互いに接する SEWER_IDS の順序リストが戻されます。

```
SELECT s1.sewer_id, s2.sewer_id
FROM sewerlines s1, sewerlines s2
WHERE db2gse.ST_Touches (s1.sewer, s2.sewer) = 1,
ORDER BY 1,2;
```

ST_Transform

ST_Transform は、図形が現在割り当てられている地理情報参照システム以外の地理情報参照システムに、図形を割り当てます。

構文

db2gse.ST_Transform(g db2gse.ST_Geometry, cr db2gse.coordref)

戻りタイプ

db2gse.ST_Geometry

例

以下の CREATE TABLE ステートメントによって TRANSFORM_TEST 表が作成されます。この表には 2 つの折れ線列 L1、L2 があります。

```
CREATE TABLE TRANSFORM_TEST (tid integer, l1 db2gse.ST_LineString,
l2 db2gse.ST_LineString)
```

以下の INSERT ステートメントは、値 102 の SRID を使って折れ線を l1 列に挿入します。

```
INSERT INTO TRANSFORM_TEST VALUES (1, db2gse.ST_LineFromText('linestring(10.01 40.43,
92.32 29.89)', db2gse.coordref()..srid(102)),NULL)
```

ST_Transform 関数は、L1 の折れ線を、SRID 102 に割り当てられた座標参照から、 SRID 105 に割り当てられた座標参照に変換します。以下の UPDATE ステートメントは、変形された折れ線を列 l2 に格納します。

```
UPDATE TRANSFORM_TEST SET l2 = db2gse.ST_Transform(l1, db2gse.coordref()..srid(105))
```

ST_Union

ST_Union は 2 つの図形オブジェクトを引き数とし、ソース・オブジェクト間の差である図形オブジェクトを戻します。

構文

db2gse.ST_Union(g1 db2gse.ST_Geometry, g2 db2gse.ST_Geometry)

戻りタイプ

db2gse.ST_Geometry

例

以下の CREATE TABLE ステートメントによって SENSITIVE_AREAS 表が作成されます。この表には、汚染の恐れがある公共施設について記述した複数の列があります。また SENSITIVE_AREAS 表には、それらの公共施設のポリゴン図形を格納する ZONE 列も含まれています。

```
CREATE TABLE SENSITIVE_AREAS (id        integer,
                              name      varchar(128),
                              size      float,
                              type      varchar(10),
                              zone      db2gse.ST_Polygon);
```

以下の CREATE TABLE ステートメントは HAZARDOUS_SITES 表を作成します。この表には、地点を識別する SITE_ID 列と NAME 列が格納されています。それぞれの地点の実際の地理上の位置は、LOCATION ポイント列に格納されています。

```
CREATE TABLE HAZARDOUS_SITES (site_id integer, name varchar(128),
location db2gse.ST_Point);
```

以下の SELECT ステートメントは、ST_Buffer 関数を使用して、危険廃棄物施設の場所を取り囲む 5 マイルの緩衝地帯を生成します。 ST_Union 関数は、危険廃棄物施設の緩衝地帯のポリゴンと重要地域との和集合からポリゴンを生成します。 ST_Area 関数は、ポリゴンの地域の和集合を戻します。

```
SELECT sa.name, hs.name,
       db2gse.ST_Area(db2gse.ST_Union(db2gse.ST_Buffer(hs.location,
(5 * 5280)),sa.zone))
FROM HAZARDOUS_SITES hs, SENSITIVE_AREAS sa;
```

ST_Within

ST_Within は 2 つの図形オブジェクトを引き数とし、最初のオブジェクトが 2 番目のオブジェクトの内部に完全に含まれていれば 1 (TRUE) を戻します。そうでなければ 0 (FALSE) を戻します。

構文

db2gse.ST_Within(g1 db2gse.ST_Geometry, g2 db2gse.ST_Geometry)

戻りタイプ

整数

例

以下の例では、2 つの表が作成されます。 1 つ目の表 BUILDINGFOOTPRINTS には都市内の建物のフットプリントが含まれています。 2 つ目の表 LOTS にはその建物の敷地が含まれています。都市計画担当者が、すべての建物のフットプリントが完全に敷地内に収まっていることを確かめたいとします。

どちらの表でも、複数ポリゴンのデータ・タイプに建物のフットプリントの図形と敷地の図形が格納されています。データベース設計担当者は、両方の図形について複数ポリゴンを選択しました。これは河川のような、地形によって敷地が分断されている場合があるため、また建物のフットプリントが複数の建物から構成されている場合があるためです。

```
CREATE TABLE BUILDINGFOOTPRINTS (   building_id integer,
                                    lot_id      integer,
                                    footprint   db2gse.ST_MultiPolygon);
CREATE TABLE LOTS (  lot_id integer, lot db2gse.ST_MultiPolygon );
```

都市計画の担当者は、以下の SELECT ステートメントを使用して、 1 つの敷地に完全には収まっていない建物を最初に選びます。

```
SELECT building_id
  FROM BUILDINGFOOTPRINTS, LOTS
 WHERE db2gse.ST_Within(footprint,lot) = 0;
```

最初の照会で敷地ポリゴンの外側にフットプリントのあるすべての BUILDING_ID がリストされるものの、残りの BUILDING_ID に lot_id が正しく割り当てられているかどうかは分かりません。以下に示す 2 番目の SELECT ステートメントは、 BUILDINGFOOTPRINTS 表の lot_id 列に対して、データ保全性の検査を実行します。

```
SELECT bf.building_id "building id",
       bf.lot_id "buildings lot_id",
       LOTS.lot_id "LOTS lot_id"
  FROM BUILDINGFOOTPRINTS bf, LOTS
 WHERE db2gse.ST_Within(footprint,lot) = 1 AND
       LOTS.lot_id <> bf.lot_id;
```

ST_WKBToSQL

ST_WKBToSQL は、事前割り当てバイナリー表現で示された ST_Geometry 値を構成します。 SRID 値 0 が自動的に使用されます。

構文

db2gse.ST_WKBToSQL(WKBGeometry Blob(1M))

戻りタイプ

db2gse.ST_Geometry

例

以下の CREATE TABLE ステートメントは LOTS 表を作成します。この表には 2 つの列があります。LOT_ID 列はそれぞれの敷地を一意的に識別し、LOT 複数ポリゴン列にはそれぞれの敷地の図形が含まれています。

```
CREATE TABLE lots (lot_id  integer,
                   lot      db2gse.ST_MultiPolygon);
```

以下の C コード断片には、 LOTS 表にデータを挿入する ODBC 関数 (DB2 地理情報エクステンダーの SQL 関数に組み込まれている) が含まれています。

ST_WKBToSQL 関数は、WKB 表現を DB2 地理情報エクステンダーの図形に変換します。 INSERT ステートメント全体は wkb_sql char string に複写されます。 INSERT ステートメントには、 LOT_ID および LOT データを動的に受け入れるパラメーター・マーカーが含まれています。

```
/* Create the SQL insert statement to populate the lot id and the
   lot multipolygon. The question marks are parameter markers that
   indicate the lot_id and lot values that will be retrieved at
   run time. */
strcpy (wkb_sql,"insert into lots (lot_id, lot)
  values(?, db2gse.ST_WKBToSQL(cast(? as blob(1m))))");
/* Allocate memory for the SQL statement handle and associate the
   statement handle with the connection handle. */
rc = SQLAllocStmt (handle, &hstmt);
/* Prepare the SQL statement for execution. */
rc = SQLPrepare (hstmt, (unsigned char *)wkb_sql, SQL_NTS);
/* Bind the integer key value to the first parameter. */
pcbvalue1 = 0;
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_SLONG,
     SQL_INTEGER, 0, 0, &lot_id, 0, &pcbvalue1);
/* Bind the shape to the second parameter. */
pcbvalue2 = blob_len;
```

```
rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_BINARY,
     SQL_BLOB, blob_len, 0, shape_blob, blob_len, &pcbvalue2);
/* Execute the insert statement. */
rc = SQLExecute (hstmt);
```

ST_WKTToSQL

ST_WKTToSQL は、事前割り当てテキスト表現で示された ST_Geometry 値を構成します。 SRID 値 0 が自動的に使用されます。

構文

db2gse.ST_WKTToSQL(geometryTaggedText Varchar(4000))

戻りタイプ

db2gse.ST_Geometry

例

以下の CREATE TABLE ステートメントによって GEOMETRY_TEST 表が作成されます。この表には 2 つの列があります。整数タイプの GID 列はそれぞれの行を一意的に識別し、G1 列には図形が格納されます。

```
CREATE TABLE GEOMETRY_TEST (gid smallint, g1 db2gse.ST_Geometry)
```

以下の INSERT ステートメントは、GEOMETRY_TEST 表の GID 列および G1 列にデータを挿入します。 ST_WKTToSQL 関数は、それぞれの図形のテキスト表現を、対応する DB2 地理情報エクステンダーのインスタンス化可能なサブクラスに変換します。

```
INSERT INTO GEOMETRY_TEST VALUES(
1, db2gse.ST_WKTToSQL ('point (10.02 20.01)')
)
INSERT INTO GEOMETRY_TEST VALUES(
2, db2gse.ST_WKTToSQL('linestring (10.01 20.01, 10.01 30.01, 10.01 40.01)')
)
INSERT INTO GEOMETRY_TEST VALUES(
3, db2gse.ST_WKTToSQL('polygon ((10.02 20.01, 11.92 35.64, 25.02 34.15, 19.15 33.94,
10.02 20.01))')
)
INSERT INTO GEOMETRY_TEST VALUES(
4, db2gse.ST_WKTToSQL('multipoint (10.02 20.01,10.32 23.98,11.92 25.64)')
)
INSERT INTO GEOMETRY_TEST VALUES(
5, db2gse.ST_WKTToSQL('multilinestring ((10.02 20.01,      10.32 23.98,11.92 25.64),
(9.55 23.75,15.36 30.11))')
)
INSERT INTO GEOMETRY_TEST VALUES(
6, db2gse.ST_WKTToSQL('multipolygon (((10.02 20.01, 11.92 35.64,
25.02 34.15, 19.15 33.94,10.02 20.01)),
((51.71 21.73, 73.36 27.04, 71.52 32.87, 52.43 31.90, 51.71 21.73)))')
)
```

ST_X

ST_X はポイントを引き数とし、その X 座標を戻します。

構文

ST_X(p ST_Point)

戻りタイプ

倍精度

例

以下の CREATE TABLE ステートメントによって X_TEST 表が作成されます。この表には、行を一意的に識別する GID 列と PT1 ポイント列の 2 つの列があります。

```
CREATE TABLE X_TEST (gid integer, pt1 db2gse.ST_Point)
```

以下の INSERT ステートメントによって 2 つの行が挿入されます。 1 つは Z 座標や測定値のないポイントです。 もう 1 つの列には Z 座標と測定値の両方があります。

```
INSERT INTO X_TEST VALUES(1,
db2gse.ST_PointFromText('point (10.02 20.01)', db2gse.coordref()..srid(0)))
INSERT INTO X_TEST VALUES(2,
db2gse.ST_PointFromText('point  zm (10.02 20.01 5.0 7.0)', db2gse.coordref()..srid(0)))
```

以下の SELECT ステートメントおよび対応する結果セットによって、 GID 列とポイントの Double X 座標がリストされます。

```
SELECT gid, db2gse.ST_X(pt1) "The X coordinate" FROM X_TEST
GID         The X coordinate
----------- ------------------------
          1   +1.00200000000000E+001
          2   +1.00200000000000E+001
  2 record(s) selected.
```

ST_Y

ST_Y はポイントを引き数とし、その Y 座標を戻します。

構文

db2gse.ST_Y(p db2gse.ST_Point)

戻りタイプ

倍精度

例

以下の CREATE TABLE ステートメントによって Y_TEST 表が作成されます。この表には、行を一意的に識別する GID 列と PT1 ポイント列の 2 つの列があります。

```
CREATE TABLE Y_TEST (gid integer, pt1 db2gse.ST_Point)
```

以下の INSERT ステートメントによって 2 つの行が挿入されます。 1 つは Z 座標や測定値のないポイントです。もう 1 つの列には Z 座標と測定値の両方があります。

```
INSERT INTO Y_TEST VALUES(1,
db2gse.ST_PointFromText('point (10.02 20.01)', db2gse.coordref()..srid(0)))
INSERT INTO Y_TEST VALUES(2,
db2gse.ST_PointFromText('point  zm (10.02 20.01 5.0 7.0)', db2gse.coordref()..srid(0)))
```

以下の SELECT ステートメントおよび対応する結果セットによって、 GID 列とポイントの Double Y 座標がリストされます。

```
SELECT gid, db2gse.ST_Y(pt1) "The Y coordinate" FROM Y_TEST
GID         The Y coordinate
----------- ------------------------
          1   +2.00100000000000E+001
          2   +2.00100000000000E+001
  2 record(s) selected.
```

Z

Z はポイントを引き数とし、その Z 座標を戻します。

構文

Z(p db2gse.ST_Point)

戻りタイプ

倍精度

例

以下の CREATE TABLE ステートメントによって Z_TEST 表が作成されます。この表には、行を一意的に識別する GID 列と PT1 ポイント列の 2 つの列があります。

```
CREATE TABLE Z_TEST (gid integer, pt1 db2gse.ST_Point)
```

以下の INSERT ステートメントによって 2 つの行が挿入されます。 1 つは Z 座標や測定値のないポイントです。もう 1 つの列には Z 座標と測定値の両方があります。

```
INSERT INTO Z_TEST VALUES(1,
db2gse.ST_PointFromText('point (10.02 20.01)', db2gse.coordref()..srid(0)))
INSERT INTO Z_TEST VALUES(2,
db2gse.ST_PointFromText('point  zm (10.02 20.01 5.0 7.0)', db2gse.coordref()..srid(0)))
```

以下の SELECT ステートメントおよび対応する結果セットによって、 GID 列とポイントの Double Z 座標がリストされます。最初の行は、ポイントに Z 座標がないので NULL となっています。

```
SELECT gid, db2gse.Z(pt1) "The Z coordinate" FROM Z_TEST
GID         The Z coordinate
----------- ------------------------
          1                        -
          2   +5.00000000000000E+000
  2 record(s) selected.
```

第15章 座標系

この章には、地理情報参照システム (SRS) と、地理情報データの解釈に使用される座標値に関する参照情報が記載されています。

座標系の概要

地理情報参照システムの事前割り当てテキスト表現では、地理情報参照システムの情報についての標準的なテキスト表現が提供されています。事前割り当てテキスト表現の定義は、 POSC/EPSG 座標系データ・モデルの後にモデル化されています。

地理情報参照システムは、地理座標系 (緯度・経度)、投影座標系 (X、Y)、または地球を中心とした座標系 (X、Y、Z) です。この座標系は、いくつかのオブジェクトで構成されています。各オブジェクトには、大文字のキーワード (たとえば、DATUM または UNIT) があり、その後に続く括弧内にはオブジェクトを定義するパラメーターがコンマで区切られて示されます。一部のオブジェクトは他のオブジェクトで構成されており、この場合、結果はネスト構造になります。

注: 実装においては、標準の括弧 ( ) を大括弧 [ ] に置き換えることができますが、両方の形式の括弧を読み込めるようにしておく必要があります。

大括弧を使用した、座標系のストリング表現に関する EBNF (拡張バッカス正規形式) 定義は次のとおりです (大括弧の使用法については上記を参照)。

```
<coordinate system> = <projected cs> | <geographic cs> | <geocentric cs>
<projected cs> = PROJCS["<name>", <geographic cs>, <projection>, {<parameter>,}*
                 <linear unit>]
<projection> = PROJECTION["<name>"]
<parameter> = PARAMETER["<name>", <value>]
<value> = <number>
```

データ・セットの座標系は、データが投影座標にある場合は PROJCS キーワードで (地理座標にある場合は GEOGCS で、地球を中心とした座標にある場合は GEOCCS で) 識別されます。 PROJCS キーワードの後には、投影座標系を定義するすべての「要素」が続きます。どのオブジェクトでも、最初の要素となるのは名前です。投影座標系の名前の後には、地理座標系、地図の投影、 1 つ以上のパラメーター、そして線形計測単位など、いくつかのオブジェクトが続きます。投影座標系のすべては、地理座標系に基づいているので、投影座標系に固有な要素についてまず説明します。たとえば、NAD83 データの UTM ゾーン 10N は、次のように定義されます。

```
PROJCS["NAD_1983_UTM_Zone_10N",
<geographic cs>,
PROJECTION["Transverse_Mercator"],
PARAMETER["False_Easting",500000.0],
PARAMETER["False_Northing",0.0],
PARAMETER["Central_Meridian",-123.0],
PARAMETER["Scale_Factor",0.9996],
PARAMETER["Latitude_of_Origin",0.0],
UNIT["Meter",1.0]]
```

この名前といくつかのオブジェクトは、地理座標系のオブジェクトをデータ、本初子午線、角度の計測単位の順に定義します。

```
<geographic cs> = GEOGCS["<name>", <datum>, <prime meridian>, <angular unit>]
<datum> = DATUM["<name>", <spheroid>]
<spheroid> = SPHEROID["<name>", <semi-major axis>, <inverse flattening>]
<semi-major axis> = <number>
    (semi-major axis is measured in meters and must be > 0.)
<inverse flattening> = <number>
<prime meridian> = PRIMEM["<name>", <longitude>]
<longitude> = <number>
```

NAD83 上の UTM ゾーン 10 の地理座標系のストリングは、次のようになります。

```
GEOGCS["GCS_North_American_1983",
DATUM["D_North_American_1983",
SPHEROID["GRS_1980",6378137,298.257222101]],
PRIMEM["Greenwich",0],
UNIT["Degree",0.0174532925199433]]
```

UNIT オブジェクトで、角度または線形の計測単位を表すことができます。

```
<angular unit> = <unit>
<linear unit> = <unit>
<unit> = UNIT["<name>", <conversion factor>]
<conversion factor> = <number>
```

変換係数には、単位あたりのメートル数 (線形の単位の場合) または単位あたりのラジアン数 (角度の単位の場合) を指定します。この値はゼロより大きくなければなりません。

それで、UTM ゾーン 10N の完全ストリング表示は、次のようになります。

```
PROJCS["NAD_1983_UTM_Zone_10N",
GEOGCS["GCS_North_American_1983",
DATUM[ "D_North_American_1983",SPHEROID["GRS_1980",6378137,298.257222101]],
PRIMEM["Greenwich",0],UNIT["Degree",0.0174532925199433]],
PROJECTION["Transverse_Mercator"],PARAMETER["False_Easting",500000.0],
PARAMETER["False_Northing",0.0],PARAMETER["Central_Meridian",-123.0],
PARAMETER["Scale_Factor",0.9996],PARAMETER["Latitude_of_Origin",0.0],
UNIT["Meter",1.0]]
```

地球を中心とした座標系は、地理座標系とよく似ています。

```
<geocentric cs> = GEOCCS["<name>", <datum>, <prime meridian>, <linear unit>]
```

サポートされている線形の単位

表 57. サポートされている線形の単位

単位	変換係数
メートル	1.0
フィート (国際)	0.3048
米国フィート	12/39.37
修正米国フィート	12.0004584/39.37
Clarke フィート	12/39.370432
Indian フィート	12/39.370141
リンク	7.92/39.370432
リンク (Benoit)	7.92/39.370113
リンク (Sears)	7.92/39.370147
チェーン (Benoit)	792/39.370113
チェーン (Sears)	792/39.370147
ヤード (Indian)	36/39.370141
ヤード (Sears)	36/39.370147
ファゾム	1.8288

表 57. サポートされている線形の単位 (続き)

単位	変換係数
海里	1852.0

サポートされている角度の単位

表 58. サポートされている角度の単位

単位	変換係数
ラジアン	1.0
10 進度	p/180
10 進分	(p/180)/60
10 進秒	(p/180)/36000
ゴン	p/200
グラジエント	p/200

サポートされている回転楕円体

表 59. サポートされている回転楕円体

名前	半主軸	逆係数
Airy	6377563.396	299.3249646
Modified Airy	6377340.189	299.3249646
Australian	6378160	298.25
Bessel	6377397.155	299.1528128
Modified Bessel	6377492.018	299.1528128
Bessel (Namibia)	6377483.865	299.1528128
Clarke 1866	6378206.4	294.9786982
Clarke 1866 (Michigan)	6378693.704	294.978684677
Clarke 1880	6378249.145	293.465
Clarke 1880 (Arc)	6378249.145	293.466307656
Clarke 1880 (Benoit)	6378300.79	293.466234571
Clarke 1880 (IGN)	6378249.2	293.46602
Clarke 1880 (RGS)	6378249.145	293.465
Clarke 1880 (SGA)	6378249.2	293.46598
Everest 1830	6377276.345	300.8017

表 59. サポートされている回転楕円体 (続き)

名前	半主軸	逆係数
Everest 1975	6377301.243	300.8017
Everest (Sarawak および Sabah)	6377298.556	300.8017
Modified Everest 1948	6377304.063	300.8017
Fischer 1960	6378166	298.3
Fischer 1968	6378150	298.3
Modified Fischer (1960)	6378155	298.3
GEM10C	6378137	298.257222101
GRS 1980	6378137	298.257222101
Hayford 1909	6378388	297.0
Helmert 1906	6378200	298.3
Hough	6378270	297.0
International 1909	6378388	297.0
International 1924	6378388	297.0
New International 1967	6378157.5	298.2496
Krasovsky	6378245	298.3
Mercury 1960	6378166	298.3
Modified Mercury 1968	6378150	298.3
NWL9D	6378145	298.25
OSU_86F	6378136.2	298.25722
OSU_91A	6378136.3	298.25722
Plessis 1817	6376523	308.64
South American 1969	6378160	298.25
Southeast Asia	6378155	298.3
球体 (半径 = 1.0)	1	0
球体 (半径 = 6371000 m)	6371000	0
球体 (半径 = 6370997 m)	6370997	0
Struve 1860	6378297	294.73
Walbeck	6376896	302.78
War Office	6378300.583	296
WGS 1960	6378165	298.3
WGS 1966	6378145	298.25

表 59. サポートされている回転楕円体 (続き)

名前	半主軸	逆係数
WGS 1972	6378135	298.26
WGS 1984	6378137	298.257223563

サポートされている測地学データ

表 60. サポートされている測地学データ

Adindan	Lisbon
Afgooye	Loma Quintana
Agadez	Lome
Australian Geodetic Datum 1966	Luzon 1911
Australian Geodetic Datum 1984	Mahe 1971
Ain el Abd 1970	Makassar
Amersfoort	Malongo 1987
Aratu	Manoca
Arc 1950	Massawa
Arc 1960	Merchich
Ancienne Triangulation Francaise	Militar-Geographische Institute
Barbados	Mhast
Batavia	Minna
Beduaram	Monte Mario
Beijing 1954	M'poraloko
Reseau National Belge 1950	NAD Michigan
Reseau National Belge 1972	North American Datum 1927
Bermuda 1957	North American Datum 1983
Bern 1898	Nahrwan 1967
Bern 1938	Naparima 1972
Bogota	Nord de Guerre
Bukit Rimpah	NGO 1948
Camacupa	Nord Sahara 1959
Campo Inchauspe	NSWC 9Z-2
Cape	Nouvelle Triangulation Francaise
Carthage	New Zealand Geodetic Datum 1949

表 60. サポートされている測地学データ (続き)

Chua	OS (SN) 1980
Conakry 1905	OSGB 1936
Corrego Alegre	OSGB 1970 (SN)
Cote d'Ivoire	Padang 1884
Datum 73	Palestine 1923
Deir ez Zor	Pointe Noire
Deutsche Hauptdreiecksnetz	Provisional South American Datum 1956
Douala	Pulkovo 1942
European Datum 1950	Qatar
European Datum 1987	Qatar 1948
Egypt 1907	Qornoq
European Reference System 1989	RT38
Fahud	South American Datum 1969
Gandajika 1970	Sapper Hill 1943
Garoua	Schwarzeck
Geocentric Datum of Australia 1994	Segora
Guyane Francaise	Serindung
Herat North	Stockholm 1938
Hito XVIII 1963	Sudan
Hu Tzu	Shan Tananarive 1925
Hungarian Datum 1972	Timbalai 1948
Indian 1954	TM65
Indian 1975	TM75
Indonesian Datum 1974	東京
Jamaica 1875	Trinidad 1903
Jamaica 1969	Trucial Coast 1948
Kalianpur	Voirol 1875
Kandawala	Voirol Unifie 1960
Kertau	WGS 1972
Kuwait Oil Company	WGS 1972 Transit Broadcast Ephemeris
La Canoa	WGS 1984
Lake	Yacare
Leigon	Yoff

表 60. サポートされている測地学データ (続き)

Liberia 1964	Zanderij

サポートされている本初子午線

表 61. サポートされている本初子午線

場所	座標
グリニッヂ	0° 0' 0"
ベルン	7° 26' 22.5" E
ボゴタ	74° 4' 51.3" W
ブリュッセル	4° 22' 4.71" E
フェロー島	17° 40' 0" W
ジャカルタ	106° 48' 27.79" E
リスボン	9° 7' 54.862" W
マドリード	3° 41' 16.58" W
パリ	2° 20' 14.025" E
ローマ	12° 27' 8.4" E
ストックホルム	18° 3' 29" E

サポートされている地図の投影法

表 62. サポートされている地図の投影法

円柱形の投影法	非円柱形の投影法
Behrmann	Craster parabolic
Cassini	Eckert I
Cylindrical equal area	Eckert II
Equirectangular	Eckert III
Gall’s stereographic	Eckert IV
Gauss-Kruger	Eckert V
Mercator	Eckert VI
Miller cylindrical	McBryde-Thomas flat polar quartic
Oblique	Mercator (Hotine) Mollweide
Plate-Carée	Robinson
Times	Sinusoidal (Sansom-Flamsteed)

表 62. サポートされている地図の投影法 (続き)

円柱形の投影法	非円柱形の投影法
Transverse Mercator	Winkel I

円すいの展開

表 63. 円すいの展開

Albers conic equal-area	Chamberlin trimetric
Bipolar oblique conformal conic	Two-point equidistant
Bonne	Hammer-Aitoff equal-area
Equidistant conic	Van der Grinten I
Lambert conformal conic	Miscellaneous
Polyconic	Alaska series E
Simple conic	Alaska Grid (Modified-Stereographic by Snyder)

方位またはプレーナーの投影法

- Azimuthal equidistant
- General vertical near-side perspective
- Gnomonic
- Lambert Azimuthal equal-area
- Orthographic
- Polar-Stereographic
- Stereographic

地図の投影のパラメーター

表 64. 地図の投影のパラメーター

パラメーター	説明
central_meridian	x 座標の原点として選択した経度の線。
scale_factor	通常は、地図の投影でのひずみの量を減らすときに使用。
standard_parallel_1	一般に、ひずみのない緯度の線。「実スケールの緯度」にも使われる。

表 64. 地図の投影のパラメーター (続き)

パラメーター	説明
standard_parallel_2	一般に、ひずみのない緯度の線。
longitude_of_center	地図の投影の中心点を定義する経度。
latitude_of_center	地図の投影の中心点を定義する緯度。
latitude_of_origin	y 座標の原点として選択した緯度。
false_easting	x 座標へ追加される。正の値を指定する際に使用。
false_northing	y 座標へ追加される。正の値を指定する際に使用。
azimuth	斜方の投影の中央線を定義する北東の角度。
longitude_of_point_1	地図の投影に必要な最初のポイントの経度。
latitude_of_point_1	地図の投影に必要な最初のポイントの緯度。
longitude_of_point_2	地図の投影に必要な 2 番目のポイントの経度。
latitude_of_point_2	地図の投影に必要な 2 番目のポイントの緯度。
longitude_of_point_3	地図の投影に必要な 3 番目のポイントの経度。
latitude_of_point_3	地図の投影に必要な 3 番目のポイントの緯度。
landsat_number	ランドサット衛星の番号。
path_number	特定の衛星の軌道の番号。
perspective_point_height	地図を投影する際の地球の観測点からの高さ。
fipszone	State Plane Coordinate System のゾーン番号。
zone	UTM のゾーン番号。

第16章 地理情報データのファイル形式

この章では、DB2 地理情報エクステンダーの事前割り当て表現について記述されています。この表現は ESRI によって定められたもので、DB2 地理情報エクステンダーに固有の表現ではないので、「*事前割り当て*」といいます。地理情報データのインポートとエクスポートを行う上で、以下の 3 種類の地理情報値について理解することが大切です。

- Open GIS Consortium (OGC) による事前割り当てテキスト表現
- OGC による事前割り当てバイナリー (WKB) 表現
- ESRI 形状表現

OGC による事前割り当てテキスト表現

DB2 地理情報エクステンダーには、テキスト記述から図形を生成する関数がいくつかあります。

ST_GeomFromText
: 任意の図形タイプのテキスト表現から図形を生成します。

ST_PointFromText
: ポイントのテキスト表現からポイントを作成します。

ST_LineFromText
: 折れ線のテキスト表現から折れ線を作成します。

ST_PolyFromText
: ポリゴンのテキスト表現からポリゴンを作成します。

ST_MPointFromText
: 複数ポイントのテキスト表現から複数ポイントを作成します。

ST_MLineFromText
: 複数折れ線のテキスト表現から複数折れ線を作成します。

ST_MPolyFromText
: 複数ポリゴンのテキスト表現から複数ポリゴンを作成します。

テキスト表現は ASCII テキスト形式のストリングです。このため、図形を ASCII テキスト形式に交換することができます。第 3 世代や第 4 世代の言語 (3GL または 4GL) のプログラムは、特殊なプログラム構造の定義が必要ない

ので、この種の関数をこれらのプログラムで使用できます。 ST_AsText 関数は、既存の図形をテキスト表現に変換します。

各図形タイプには、事前割り当てのテキスト表現が含まれており、新しいインスタンスのタイプを構成する場合と、既存のインスタンスを英数字表示のためのテキスト表現に変換する場合の両方に使用することができます。

図形の事前割り当てテキスト表現を以下に定義します。 {}* は、括弧内のトークンを 0 回以上繰り返すことを示します。出力トークンのリストには、括弧は示されません。

```
<Geometry Tagged Text> :=
| <Point Tagged Text>
| <LineString Tagged Text>
| <Polygon Tagged Text>
| <MultiPoint Tagged  Text>
| <MultiLineString Tagged Text>
| <MultiPolygon Tagged Text>

<Point Tagged Text> :=
POINT <Point Text>

<LineString Tagged Text> :=
LINESTRING <LineString Text>

<Polygon Tagged Text> :=
POLYGON <Polygon Text>

<MultiPoint Tagged Text> :=
MULTIPOINT <Multipoint Text>

<MultiLineString Tagged Text> :=
MULTILINESTRING <MultiLineString Text>

<MultiPolygon Tagged Text> :=
MULTIPOLYGON <MultiPolygon Text>

<Point Text> := EMPTY
|    <Point>
| Z  <PointZ>
| M  <PointM>
| ZM <PointZM>

<Point> :=  <x>  <y>
<x> := double precision literal
<y> := double precision literal
<PointZ> :=  <x>  <y>  <z>
<x> := double precision literal
<y> := double precision literal
<z> := double precision literal
<PointM> :=  <x>  <y>  <m>
<x> := double precision literal
<y> := double precision literal
```

```
<m> := double precision literal
<PointZM> :=  <x>  <y>  <z>  <m>
<x> := double precision literal
<y> := double precision literal
<z> := double precision literal
<m> := double precision literal

<LineString Text> := EMPTY
|    ( <Point Text >   {,  <Point Text> }*  )
| Z  ( <PointZ Text >  {,  <PointZ Text> }*  )
| M  ( <PointM Text >  {,  <PointM Text> }*  )
| ZM ( <PointZM Text > {,  <PointZM Text> }*  )

<Polygon Text> := EMPTY
| ( <LineString Text > {,< LineString Text > }*  )

<Multipoint Text> := EMPTY
| ( <Point Text >   {,  <Point Text > }*  )

<MultiLineString Text> := EMPTY
| ( <LineString Text > {,< LineString Text>}*  )

<MultiPolygon Text> := EMPTY
| ( < Polygon Text > {,  < Polygon Text > }*  )
```

基本的な関数の構文は次のとおりです。

```
function (<text description>,<SRID>)
```

SRID は、SPATIAL_REFERENCES 表の地理情報参照 ID かつ基本キーであり、 SPATIAL_REFERENCES 表に格納されている図形の地理情報参照システムを識別します。図形を地理情報列に挿入する場合、図形の SRID が地理情報列の SRID と一致しなければなりません。

テキスト記述は、以下の例のように、単一引用符で囲まれた 3 つの基本的な構成要素で構成されています。

```
<'geometry type'> ['coordinate type'] [''coordinate list']
```

ここで、

geometry type
: ポイント、折れ線、ポリゴン、複数ポイント、複数折れ線、または複数ポリゴンのいずれか。

coordinate type
: 図形に Z 座標または測定値を含めるかどうかを指定します。図形にどちらもない場合は、この引き数をブランクにしておいてください。そう

でない場合、coordinate type は、Z 座標を含む図形については Z に、測定値を含む図形については M に、両方を含む図形については ZM に設定してください。

coordinate list
図形の頂点を定義します。座標リストはコンマで区切られており、括弧で囲まれています。複数の構成要素を含む図形には、各構成要素の部分を囲むために、一連の括弧が必要です。図形が空である場合、EMPTY キーワードによって座標を置き換えます。

表65 には、使用できるすべてのテキスト表現の例が漏れなくリストされています。

表 65. 図形タイプとそのテキスト表現

図形タイプ	テキスト記述	注釈
ポイント	point empty	空のポイント
ポイント	point z empty	z 座標を持つ空のポイント
ポイント	point m empty	測定値を持つ空のポイント
ポイント	point zm empty	z 座標と測定値を持つ空のポイント
ポイント	point (10.05 10.28)	ポイント
ポイント	point z (10.05 10.28 2.51)	z 座標を持つポイント
ポイント	point m (10.05 10.28 4.72)	測定値を持つポイント
ポイント	point zm (10.05 10.28 2.51 4.72)	z 座標と測定値を持つポイント
折れ線	linestring empty	空の折れ線
折れ線	linestring z empty	z 座標を持つ空の折れ線
折れ線	linestring m empty	測定値を持つ空の折れ線
折れ線	linestring zm empty	z 座標と測定値を持つ空の折れ線
折れ線	linestring (10.05 10.28 , 20.95 20.89)	折れ線
折れ線	linestring z (10.05 10.28 3.09, 20.95 31.98 4.72, 21.98 29.80 3.51)	z 座標を持つ折れ線
折れ線	linestring m (10.05 10.28 5.84, 20.95 31.98 9.01, 21.98 29.80 12.84)	測定値を持つ折れ線

表 65. 図形タイプとそのテキスト表現 (続き)

図形タイプ	テキスト記述	注釈
折れ線	linestring zm ( )	z 座標と測定値を持つ折れ線
ポリゴン	polygon empty	空のポリゴン
ポリゴン	polygon z empty	z 座標を持つ空のポリゴン
ポリゴン	polygon m empty	測定値を持つポリゴン
ポリゴン	polygon zm empty	z 座標と測定値を持つ空のポリゴン
ポリゴン	polygon ((10 10, 10 20, 20 20, 20 15, 10 10))	ポリゴン
ポリゴン	polygon z (( ))	z 座標を持つポリゴン
ポリゴン	polygon m (( ))	測定値を持つポリゴン
ポリゴン	polygon zm (( ))	z 座標と測定値を持つポリゴン
複数ポイント	multipoint empty	空の複数ポイント
複数ポイント	multipoint z empty	z 座標を持つ空の複数ポイント
複数ポイント	multipoint m empty	測定値を持つ空の複数ポイント
複数ポイント	multipoint zm empty	z 座標と測定値を持つ空の複数ポイント
複数ポイント	multipoint empty	空の複数ポイント
複数ポイント	multipoint (10 10, 20 20)	2 つのポイントを持つ複数ポイント
複数ポイント	multipoint z (10 10 2, 20 20 3)	z 座標を持つ複数ポイント
複数ポイント	multipoint m (10 10 4, 20 20 5)	測定値を持つ複数ポイント
複数ポイント	multipoint zm (10 10 2 4, 20 20 3 5)	z 座標と測定値を持つ複数ポイント
複数折れ線	multilinestring empty	空の複数折れ線
複数折れ線	multilinestring z empty	z 座標を持つ空の複数折れ線
複数折れ線	multilinestring m empty	測定値を持つ空の複数折れ線

表 65. 図形タイプとそのテキスト表現 (続き)

図形タイプ	テキスト記述	注釈
複数折れ線	multilinestring zm empty	z 座標と測定値を持つ複数折れ線
複数折れ線	multilinestring (( ))	複数折れ線
複数折れ線	multilinestring z (( ))	z 座標を持つ複数折れ線
複数折れ線	multilinestring m (( ))	測定値を持つ複数折れ線
複数折れ線	multilinestring zm (( ))	z 座標と測定値を持つ複数折れ線
複数ポリゴン	multipolygon empty	空の複数ポリゴン
複数ポリゴン	multipolygon z empty	z 座標を持つ空の複数ポリゴン
複数ポリゴン	multipolygon m empty	測定値を持つ空の複数ポリゴン
複数ポリゴン	multipolygon z	z 座標と測定値を持つ複数ポリゴン
複数ポリゴン	multipolygon ((( )))	複数ポリゴン
複数ポリゴン	multipolygon z ((( )))	z 座標を持つ複数ポリゴン
複数ポリゴン	multipolygon m (((10 10 2, 10 20 3, 20 20 4, 20 15 5, 10 10 2), (50 40 7, 50 50 3, 60 50 4, 60 40 5, 50 40 7)))	測定値を持つ複数ポリゴン
複数ポリゴン	multipolygon zm ((( )))	z 座標と測定値を持つ複数ポリゴン

OGC による事前割り当てバイナリー (WKB) 表現

DB2 地理情報エクステンダーには、バイナリー表現から図形を生成する関数がいくつかあります。

ST_GeomFromWKB
任意の図形タイプの WKB 表現から図形を作成します。

ST_PointFromWKB
ポイントの WKB 表現からポイントを作成します。

ST_LineFromWKB
折れ線の WKB 表現から折れ線を作成します。

ST_PolyFromWKB
ポリゴンの WKB 表現からポリゴンを作成します。

ST_MPointFromWKB
複数ポイントの WKB 表示から複数ポイントを作成します。

ST_MLineFromWKB
複数折れ線の WKB 表現から複数折れ線を作成します。

ST_MPolyFromWKB
複数ポリゴンの WKB 表現から複数ポリゴンを作成します。

事前割り当てバイナリー表現は、連続する一連のバイトです。これにより、ODBC クライアントと SQL データベースの間で図形をバイナリー形式で交換することができます。これらの図形関数では、バイナリー表現をマップする C プログラミング言語構造を定義する必要があるので、第 3 世代 (3GL) プログラムで使用するよう意図されています。第 4 世代言語 (4GL) 環境での使用には適していません。 ST_AsBinary 関数は、既存の図形値を事前割り当てバイナリー表現に変換します。

図形の事前割り当てバイナリー表現を入手するには、数値型順序列として図形インスタンスを順序付けします。このタイプはセット (Unsigned Integer, Double) から引き出され、その後、それぞれの数値型をバイトの順序列として順序付けします。タイプの順序付けには、数値型を表す標準の定義済みバイナリー表現 (NDR、XDR の 2 つがある) のいずれかを使います。順序付けされたバイトに先行する 1 バイトのタグとして、図形バイト・ストリームに使われる特定のバイナリー・エンコード (NDR または XDR) が記述されます。この 2 つの形状のエンコードの唯一の違いは、バイト順序の違いです。 XDR エンコードはビッグ・エンディアンであり、 NDR エンコードはリトル・エンディアンです。

数値型の定義

無符号整数 は、 [0, 4294967295] の範囲にある負でない整数をコード化する 32 ビット (4 バイト) のデータ型です。

倍精度 は、 IEEE 754 倍精度形式を用いて倍精度の数値をコード化する 64 ビット (8 バイト) の倍精度データ型です。

これらの定義は、XDR、NDR の両方に共通です。

数値型の XDR (ビッグ・エンディアン) エンコード

無符号整数の XDR 表現はビッグ・エンディアンです (最初のバイトが最上位桁)。

倍精度の XDR 表現はビッグ・エンディアンです (最初のバイトが符号ビット)。

数値型の NDR (リトル・エンディアン) エンコード

無符号整数の NDR 表現はリトル・エンディアンです (最初のバイトが最下位桁)。

倍精度の NDR 表現はリトル・エンディアンです (最後のバイトが符号ビット)。

NDR と XDR の間の変換

NDR データ型と XDR データ型の間の無符号整数と倍精度の変換は、簡単な操作です。この操作には、バイト・ストリーム内にあるそれぞれの無符号整数または倍精度でバイトの順序を反転させることが関係します。

WKBGeometry バイト・ストリームの説明

図形の事前割り当てバイナリー表現について以下に説明します。基本的な構成ブロックは、2 つの倍精度からなるポイントのバイト・ストリームです。ほかの図形のバイト・ストリームは、定義済みの図形のバイト・ストリームを使って作成されます。

```
// Basic Type definitions
// byte : 1 byte
// uint32 : 32 bit unsigned integer  (4 bytes)
// double : double precision number (8 bytes)

// Building Blocks : Point, LinearRing

Point {
  double x;
  double y;
};
LinearRing   {
  uint32  numPoints;
  Point   points[numPoints];
};
enum wkbGeometryType {
  wkbPoint = 1,
  wkbLineString = 2,
  wkbPolygon = 3,
  wkbMultiPoint = 4,
  wkbMultiLineString = 5,
```

```
  wkbMultiPolygon = 6,
};
enum wkbByteOrder {
  wkbXDR = 0,                                    // Big Endian
  wkbNDR = 1                                   // Little Endian
};
WKBPoint {
  byte     byteOrder;
  uint32   wkbType;                                        // 1
  Point    point;
};
WKBLineString {
  byte     byteOrder;
  uint32   wkbType;                                        // 2
  uint32   numPoints;
  Point   points[numPoints];
}

WKBPolygon    {
  byte                byteOrder;
  uint32            wkbType;                               // 3
  uint32            numRings;
  LinearRing        rings[numRings];
}
WKBMultiPoint    {
  byte                byteOrder;
  uint32            wkbType;                               // 4
  uint32            num_wkbPoints;
  WKBPoint            WKBPoints[num_wkbPoints];
}
WKBMultiLineString    {
  byte              byteOrder;
  uint32            wkbType;                               // 5
  uint32            num_wkbLineStrings;
  WKBLineString     WKBLineStrings[num_wkbLineStrings];
}

wkbMultiPolygon {
  byte              byteOrder;
  uint32            wkbType;                               // 6
  uint32            num_wkbPolygons;
  WKBPolygon        wkbPolygons[num_wkbPolygons];
}

WKBGeometry  {
  union {
    WKBPoint                point;
    WKBLineString           linestring;
    WKBPolygon              polygon;
    WKBMultiPoint           mpoint;
    WKBMultiLineString      mlinestring;
    WKBMultiPolygon         mpolygon;
  }
};
```

以下の図は、NDR 表現を示しています。

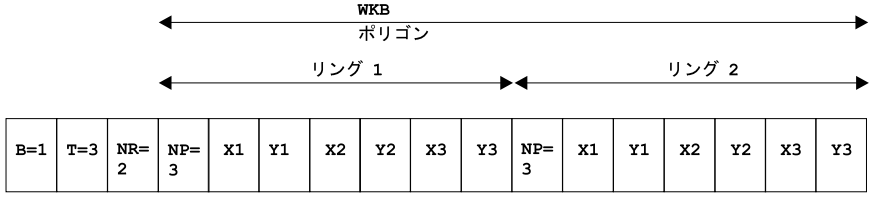


図 39. NDR フォーマットでの表示. (B=1) タイプのポリゴン (T=3) には 2 つのリニア (NR=2) があり、それぞれのリングには 3 つのポイント (NP=3) がある。

WKB 表現での宣言

図形の事前割り当てバイナリー表現は、 Geometry Object Model および OpenGIS Abstract Specification で記述されている図形タイプのインスタンスを表すように設計されています。

これらの宣言では、リング、ポリゴンおよび複数ポリゴンについて、以下のように暗黙指定されています。

線形リング

リングは単純で閉じています。つまり線形リングは自身と交差できません。

ポリゴン

任意のポリゴンの境界にある線形リングのうち、 2 つが互いに重なり合うことはできません。ポリゴンの境界にある線形リングは、交差できるとしても単一のポイントで接するだけです。

複数ポリゴン

複数ポリゴンの要素であるポリゴンのうち、内部で 2 つが互いに交差することはできません。複数ポリゴンの要素である任意の 2 つのポリゴンの境界は、接することができる場合でも接点の数は有限です。

ESRI 形状表現

DB2 地理情報エクステンダーには、ESRI 形状表現から図形を生成する関数がいくつかあります。 Open GIS 事前割り当てバイナリー表現によってサポートされる 2 次元表現に加えて、 ESRI 形状表現もオプションの Z 座標と測定値をサポートしています。以下の関数は、ESRI 形状から図形を生成します。

ST_GeometryFromShape
任意の図形タイプの形状表現から図形を作成します。

ST_PointFromShape
ポイントの形状表現からポイントを作成します。

ST_LineFromShape
折れ線の形状表現から折れ線を作成します。

ST_PolyFromShape
ポリゴンの形状表現からポリゴンを作成します。

ST_MPointFromShape
複数ポイントの形状表示から複数ポイントを作成します。

ST_MLineFromShape
複数折れ線の形状表現から複数折れ線を作成します。

ST_MPolyFromShape
複数ポリゴンの形状表現から複数ポリゴンを作成します。

これらの関数の一般的な構文は同じです。最初の引き数は、2 進ラージ・オブジェクト (BLOB) データ・タイプとして入力される形状表現です。 2 番目の引き数は、図形に割り当てる地理情報参照 ID の整数です。
GeometryFromShape 関数は以下のような構文になります。

```
db2gse.GeometryFromShape(ShapeGeometry Blob(1M), cr db2gse.coordref)
```

これらの形状関数では、バイナリー表現をマップする C プログラミング言語構造を定義する必要があるので、 3GL プログラムで使用するよう意図されており、4GL 環境での使用には適していません。 AsShape 関数は、図形値を ESRI 形状表現に変換します。

0 という形状タイプは、形状の図形データがない、ヌル形状を示しています。

値	形状タイプ
0	ヌル形状
1	ポイント
3*	折れ線
5	ポリゴン
8	複数ポイント
11	ポイント Z
13	折れ線 Z
15	ポリゴン Z

値	形状タイプ
18	複数ポイント Z
21	ポイント M
23	折れ線 M
25	ポリゴン M
28	複数ポイント M

注: * 上記で示されていない形態タイプ (2、4、6、など) は、将来の利用のために予約済みです。

XY スペースでの形状タイプ

ポイント

ポイントは、X、Y という順番の、一組の倍精度の座標で構成されています。

表 66. ポイントのバイト・ストリームの内容

位置	フィールド	値	タイプ	数	順序
バイト 0	形状タイプ	1	整数	1	リトル
バイト 4	X	X	倍精度	1	リトル
バイト 12	Y	Y	倍精度	1	リトル

複数ポイント

複数ポイントは、ポイントの集合で構成されています。境界ボックスは、Xmin、Ymin、Xmax、Ymax の順序で保管されています。

表 67. 複数ポイントのバイト・ストリームの内容

位置	フィールド	値	タイプ	数	順序
バイト 0	形状タイプ	8	整数	1	リトル
バイト 4	Box	Box	倍精度	4	リトル
バイト 36	NumPoints	NumPoints	整数	1	リトル
バイト 40	Points	Points	ポイント	NumPoints	リトル

折れ線

折れ線は、1 つまたは複数のパーツから成る、順序が付けられた頂点の集合です。パーツは、一連の複数のポイントをつなげたものです。ポイントはお互いにつながっている場合と、そうでない場合があります。パーツはお互いに交差している場合と、そうでない場合があります。

この仕様では連続するポイントが同一の座標を持つことを禁止していないので、形状ファイル読み取りプログラムはこのような場合も処理しなければなりません。他方、結果としてパーツが短くなり、長さが 0 になることは、許可されません。

以下が折れ線のフィールドです。

Box Xmin、Ymin、Xmax、Ymax の順序で保管されている、折れ線の境界ボックス。

NumParts
折れ線にあるパーツの数。

NumPoints
すべてのパーツにあるポイントの合計数。

Parts NumParts の長さの配列。折れ線ごとに、ポイントの配列における最初のポイントの指数を保管します。配列指数は、0 をベースとしています。

Points

NumPoints の長さの配列。折れ線にある各パーツのポイントを端から端まで保管します。パーツ 1 のポイントの後にパーツ 2 のポイントが続くというようにつなげます。パーツの配列には、パーツごとの最初のポイントの配列指数が保持されます。パーツ間のポイントの配列には、区切り文字は入りません。

表 68. 折れ線のバイト・ストリームの内容

位置	フィールド	値	タイプ	数	順序
バイト 0	形状タイプ	3	整数	1	リトル
バイト 4	Box	Box	倍精度	4	リトル
バイト 36	NumParts	NumParts	整数	1	リトル
バイト 40	NumPoints	NumPoints	整数	1	リトル
バイト 44	Parts	Parts	整数	NumParts	リトル
バイト X	Points	Points	ポイント	NumPoints	リトル

注: X = 44 + 4 * NumParts

ポリゴン

ポリゴンは 1 つまたは複数のリングで構成されています。リングとは、閉じた形の 4 つ以上の連続するポイントをつなげたもので、自分自身と交差することのないループになっています。ポリゴンには、複数の外部リングを含めること

ができます。頂点の順序やリングの方向は、リングのどちらの側がポリゴンの内部であるかを示します。頂点の順序でリングにそって歩くとすると、右側がポリゴンの内側になります。ポリゴンの中の穴を定義するリングの頂点は、左回りの順序になっています。そのため、1 つのリングであるポリゴンの頂点は、必ず右回りの順序になります。ポリゴンのリングのことを、そのポリゴンのパーツといいます。

この仕様では連続するポイントが同一の座標を持つことを禁止していないので、形状ファイル読み取りプログラムはこのような場合も処理しなければなりません。他方、結果としてパーツが小さくなり、長さや面積が 0 になることは、許可されません。

以下がポリゴンのフィールドです。

Box Xmin、Ymin、Xmax、Ymax の順序で保管されている、ポリゴンの境界ボックス。

NumParts
ポリゴンのリングの数。

NumPoints
すべてのリングにあるポイントの合計数。

Parts NumParts の長さの配列。リングごとに、ポイントの配列における最初のポイントの指数を保管します。配列指数は、0 をベースとしています。

Points

NumPoints の長さの配列。ポリゴンにある各リングのポイントを端から端まで保管します。リング 1 のポイントの後にリング 2 のポイントが続くというようにつなげます。パーツの配列には、リングごとの最初のポイントの配列指数が保持されます。リング間のポイントの配列には、区切り文字は入りません。

ポリゴンの形状に関する重要な注釈

- リングは閉じています (リングの最初と最後の頂点は必ず同じでなければなりません)。
- ポイントの配列におけるリングの順序は重要ではありません。
- 形状ファイルに保管されるポリゴンは、きちんとしたものでなければなりません。きちんとしたポリゴンとは、以下のようなものです。
  - 自分自身と交差していない。つまり、あるリングに属する区画が、別のリングに属する区画と交差することはできません。ポリ

ゴンのリングはお互いに頂点で接することができますが、区画にそって交差することはできません。同一直線上にある区画は、交差していると見なされます。

– 定義している線の「正しい」側にポリゴンの内側がある。頂点の順序でリングにそって歩くとすると、右側がポリゴンの内側になります。そのため、1 つのリングであるポリゴンの頂点は、必ず右回りの順序になります。これらのポリゴンにある穴を定義するリングは、左回りの方向になります。

「きちんとしていない」ポリゴンは、ポリゴン内の穴を定義するリングも右回りになっている場合に生じます。この場合、内部が重なり合っています。

ポリゴンのインスタンスの例:

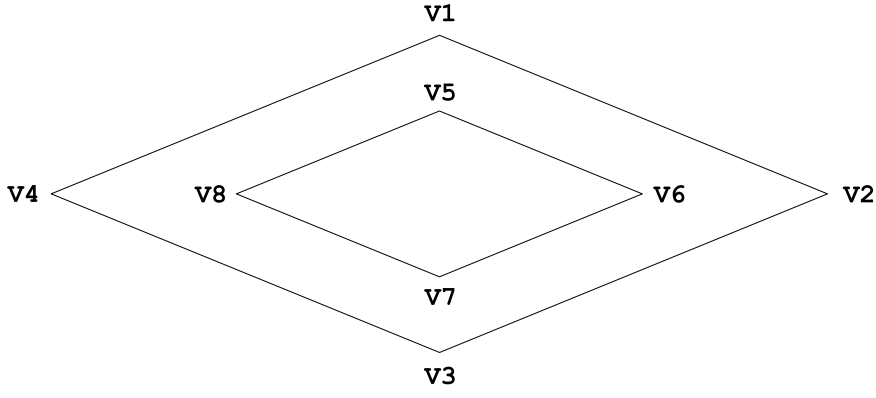


図 40. 1 つの穴と 8 つの頂点があるポリゴン

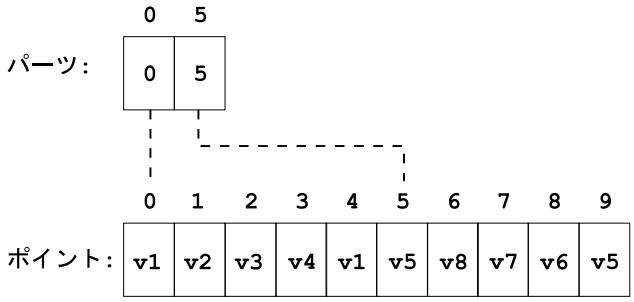


図 41. ポリゴンのバイト・ストリームの内容. NumParts は 2 に等しく、NumPoints は 10 に等しくなっています。ドーナツ (穴) ポリゴンにあるポイントの順序が逆になっていることに注意してください。

表 69. ポリゴンのバイト・ストリームの内容

位置	フィールド	値	タイプ	数	順序
バイト 0	形状タイプ	5	整数	1	リトル
バイト 4	Box	Box	倍精度	4	リトル
バイト 36	NumParts	NumParts	整数	1	リトル
バイト 40	NumPoints	NumPoints	整数	1	リトル
バイト 44	Parts	Parts	整数	NumParts	リトル
バイト X	Points	Points	ポイント	NumPoints	リトル

注: X = 44 + 4 * NumParts

XY スペースでの測定された形状タイプ

ポイント M

ポイント M は、X、Y という順番の一組の倍精度の座標と測定値 M で構成されています。

表 70. ポイント M のバイト・ストリームの内容

位置	フィールド	値	タイプ	数	順序
バイト 0	形状タイプ	21	整数	1	リトル
バイト 4	X	X	倍精度	1	リトル
バイト 12	Y	Y	倍精度	1	リトル
バイト 20	M	M	倍精度	1	リトル

複数ポイント M

以下が複数ポイント M のフィールドです。

Box Xmin、Ymin、Xmax、Ymax の順番で保管されている、複数ポイント M の境界ボックス。

NumPoints

ポイントの数。

Points

NumPoints の長さのポイントの配列。

NumMs

後に続く測定値の数。このフィールドの後に測定値が入力されていない場合、 NumMs は 2 つのゼロ値しかとることができません。測定値が入力されている場合は、NumPoints と等しくなります。

M Range

Mmin、Mmax の順番で保管される、複数ポイント M の最小と最大の測定値。

M Array

NumPoints の長さの測定値の配列。

表 71. 複数ポイント M のバイト・ストリームの内容

位置	フィールド	値	タイプ	数	順序
バイト 0	形状タイプ	28	整数	1	リトル
バイト 4	Box	Box	倍精度	4	リトル
バイト 36	NumPoints	NumPoints	整数	1	リトル
バイト 40	Points	Points	ポイント	NumPoints	リトル
バイト X	NumMs	NumMs	整数	1	リトル
バイト X+4*	Mmin	Mmin	倍精度	1	リトル
バイト X+12*	Mmax	Mmax	倍精度	1	リトル
バイト X+20*	Marray	Marray	倍精度	NumPoints	リトル

注:

1. X = 40 + (16 * NumPoints)
2. * 任意指定

折れ線 M

形状ファイルである折れ線 M は、1 つまたは複数のパーツで構成されています。パーツは、一連の複数のポイントをつなげたものです。パーツはお互いにつながっている場合と、そうでない場合があります。また、お互いに交差している場合と、そうでない場合があります。

以下が折れ線 M のフィールドです。

Box Xmin、Ymin、Xmax、Ymax の順番で保管されている、折れ線 M の境界ボックス。

NumParts
折れ線 M にあるパーツの数。

NumPoints
すべてのパーツにあるポイントの合計数。

Parts NumParts の長さの配列。パーツごとに、ポイントの配列における最初のポイントの指数を保管します。配列指数は、0 をベースとしています。

Points
NumPoints の長さの配列。折れ線 M にある各パーツのポイントを端から端まで保管します。パーツ 1 のポイントの後にパーツ 2 のポイントが続くというようにつなげます。パーツの配列には、パーツごとの最初のポイントの配列指数が保持されます。パーツ間のポイントの配列には、区切り文字は入りません。

NumMs
後に続く測定値の数。このフィールドの後に測定値が入力されていない場合、 NumMs は 2 つのゼロ値しかとることができません。測定値が入力されている場合は、NumPoints と等しくなります。

M Range
Mmin、Mmax の順番で保管される、折れ線 M の最小と最大の測定値。

M Array
NumPoints の長さの配列。折れ線 M にある各パーツの測定値を端から端まで保管します。パーツ 1 の測定値の後にパーツ 2 の測定値が続くようにつなげます。パーツの配列には、パーツごとの最初のポイントの配列指数が保持されます。パーツ間の測定値の配列には、区切り文字は入りません。

表 72. 折れ線 M のバイト・ストリームの内容

位置	フィールド	値	タイプ	数	順序
バイト 0	形状タイプ	13	整数	1	リトル
バイト 4	Box	Box	倍精度	4	リトル
バイト 36	NumParts	NumParts	整数	1	リトル
バイト 40	NumPoints	NumPoints	整数	1	リトル
バイト 44	Parts	Parts	整数	NumParts	リトル
バイト X	Points	Points	ポイント	NumPoints	リトル
バイト Y	NumMs	NumMs	整数	1	リトル
バイト Y+4*	Mmin	Mmin	倍精度	1	リトル
バイト Y+12*	Mmax	Mmax	倍精度	1	リトル
バイト Y+20*	Marray	Marray	倍精度	NumPoints	リトル

注:

1. X = 44 + (4 * NumParts)、Y = X + (16 * NumPoints)
2. * 任意指定

ポリゴン M

ポリゴン M は、多数のリングで構成されています。リングは、閉じた形で自分自身と交差することのないループになっています。交差部分が計算されるのは XY スペースであって、 XYM スペースではない ことに注意してください。ポリゴン M には、複数の外部リングを含めることができます。ポリゴン M のリングのことを、そのポリゴン M のパーツといいます。

以下がポリゴン M のフィールドです。

Box Xmin、Ymin、Xmax、Ymax の順番で保管されている、ポリゴン M の境界ボックス。

NumParts
ポリゴン M のリングの数。

NumPoints
すべてのリングにあるポイントの合計数。

Parts NumParts の長さの配列。リングごとに、ポイントの配列における最初のポイントの指数を保管します。配列指数は、0 をベースとしています。

Points

NumPoints の長さの配列。ポリゴン M にある各リングのポイントを端から端まで保管します。リング 1 のポイントの後にリング 2 のポイントが続くというようにつなげます。パーツの配列には、リングごとの最初のポイントの配列指数が保持されます。リング間のポイントの配列には、区切り文字は入りません。

NumMs

後に続く測定値の数。このフィールドの後に測定値が入力されていない場合、 NumMs は 2 つのゼロ値しかとることができません。測定値が入力されている場合は、NumPoints と等しくなります。

M Range

Mmin、Mmax の順番で保管される、ポリゴン M の最小と最大の測定値。

M Array

NumPoints の長さの配列。ポリゴン M にある各リングの測定値を端から端まで保管します。リング 1 の測定値の後にリング 2 の測定値が続くというようにつなげます。パーツの配列には、リングごとの最初の測定値の配列指数が保持されます。リング間の測定値の配列には、区切り文字は入りません。

ポリゴン M の形状に関する重要な注釈

- リングは閉じています (リングの最初と最後の頂点は必ず同じでなければなりません)。
- ポイントの配列におけるリングの順序は重要ではありません。

表 73. ポリゴン M のバイト・ストリームの内容

位置	フィールド	値	タイプ	数	順序
バイト 0	形状タイプ	15	整数	1	リトル
バイト 4	Box	Box	倍精度	4	リトル
バイト 36	NumParts	NumParts	整数	1	リトル
バイト 40	NumPoints	NumPoints	整数	1	リトル
バイト 44	Parts	Parts	整数	NumParts	リトル
バイト X	Points	Points	ポイント	NumPoints	リトル
バイト Y	NumMs	NumMs	整数	1	リトル
バイト Y+4*	Mmin	Mmin	倍精度	1	リトル
バイト Y+12*	Mmax	Mmax	倍精度	1	リトル

表 73. ポリゴン M のバイト・ストリームの内容 (続き)

位置	フィールド	値	タイプ	数	順序
バイト Y+20*	Marray	Marray	倍精度	NumPoints	リトル

注:

1. X = 44 + (4 * NumParts)、Y = X + (16 * NumPoints)
2. * 任意指定

XYZ スペースでの形状タイプ

ポイント Z

ポイント Z は、X、Y、Z という順番の三つ組の倍精度の座標と測定値で構成されています。

表 74. ポイント Z のバイト・ストリームの内容

位置	フィールド	値	タイプ	数	順序
バイト 0	形状タイプ	11	整数	1	リトル
バイト 4	X	X	倍精度	1	リトル
バイト 12	Y	Y	倍精度	1	リトル
バイト 20	Z	Z	倍精度	1	リトル
バイト 28	測定値	M	倍精度	1	リトル

複数ポイント Z

複数ポイント Z は、以下のようなポイント Z の集合を表しています。

- 境界ボックスは、Xmin、Ymin、Xmax、Ymax の順序で保管されています。
- 境界となる Z 範囲は、Zmin、Zmax の順番で保管されています。境界となる M Range は、Mmin、Mmax の順番で保管されています。

表 75. 複数ポイント Z のバイト・ストリームの内容

位置	フィールド	値	タイプ	数	順序
バイト 0	形状タイプ	18	整数	1	リトル
バイト 4	Box	Box	倍精度	4	リトル
バイト 36	NumPoints	NumPoints	整数	1	リトル
バイト 40	Points	Points	ポイント	NumPoints	リトル
バイト X	Zmin	Zmin	倍精度	1	リトル
バイト X+8	Zmax	Zmax	倍精度	1	リトル

表 75. 複数ポイント Z のバイト・ストリームの内容 (続き)

位置	フィールド	値	タイプ	数	順序
バイト X+16	Zarray	Zarray	倍精度	NumPoints	リトル
バイト Y	NumMs	NumMs	整数	1	リトル
バイト Y+4*	Mmin	Mmin	倍精度	1	リトル
バイト Y+12*	Mmax	Mmax	倍精度	1	リトル
バイト Y+20*	Marray	Marray	倍精度	NumPoints	リトル

注:

1. X = 40 + (16 * NumPoints)、Y = X + 16 + (8 * NumPoints)
2. * 任意指定

折れ線 Z

折れ線 Z は 1 つまたは複数のパーツで構成されています。パーツは、一連の複数のポイントをつなげたものです。パーツはお互いにつながっている場合と、そうでない場合があります。また、お互いに交差している場合と、そうでない場合があります。

以下が折れ線 Z のフィールドです。

Box Xmin、Ymin、Xmax、Ymax の順番で保管されている、折れ線 Z の境界ボックス。

NumParts
折れ線 Z にあるパーツの数。

NumPoints
すべてのパーツにあるポイントの合計数。

Parts NumParts の長さの配列。パーツごとに、ポイントの配列における最初のポイントの指数を保管します。配列指数は、0 をベースとしています。

Points
NumPoints の長さの配列。折れ線 Z にある各パーツのポイントを端から端まで保管します。パーツ 1 のポイントの後にパーツ 2 のポイントが続くというようにつなげます。パーツの配列には、パーツごとの最初のポイントの配列指数が保持されます。パーツ間のポイントの配列には、区切り文字は入りません。

Z Range

Zmin、Zmax の順序で保管される、折れ線 Z の最小と最大の Z 値。

Z Array

NumPoints の長さの配列。折れ線 Z にある各パーツの Z 値を端から端まで保管します。パーツ 1 の Z 値の後にパーツ 2 の Z 値が続くようにつなげます。パーツの配列には、パーツごとの最初のポイントの配列指数が保持されます。パーツ間の Z 値の配列には、区切り文字は入りません。

NumMs

後に続く測定値の数。このフィールドの後に測定値が入力されていない場合、 NumMs は 2 つのゼロ値しかとることができません。測定値が入力されている場合は、NumPoints と等しくなります。

M Range

Mmin、Mmax の順番で保管される、折れ線 Z の最小と最大の測定値。

M Array

NumPoints の長さの配列。折れ線 Z にある各パーツの測定値を端から端まで保管します。パーツ 1 の測定値の後にパーツ 2 の測定値が続くようにつなげます。パーツの配列には、パーツごとの最初の測定値の配列指数が保持されます。パーツ間の測定値の配列には、区切り文字は入りません。

表 76. 折れ線 Z のバイト・ストリームの内容

位置	フィールド	値	タイプ	数	順序
バイト 0	形状タイプ	13	整数	1	リトル
バイト 4	Box	Box	倍精度	4	リトル
バイト 36	NumParts	NumParts	整数	1	リトル
バイト 40	NumPoints	NumPoints	整数	1	リトル
バイト 44	Parts	Parts	整数	NumParts	リトル
バイト X	Points	Points	ポイント	NumPoints	リトル
バイト Y	Zmin	Zmin	倍精度	1	リトル
バイト Y+8	Zmax	Zmax	倍精度	1	リトル
バイト Y+16	Zarray	Zarray	倍精度	NumPoints	リトル
バイト Z	NumMs	NumMs	整数	1	リトル
バイト Z+4*	Mmin	Mmin	倍精度	1	リトル

表 76. 折れ線 Z のバイト・ストリームの内容 (続き)

位置	フィールド	値	タイプ	数	順序
バイト Z+12*	Mmax	Mmax	倍精度	1	リトル
バイト Z+20*	Marray	Marray	倍精度	NumPoints	リトル

注:

1. X = 44 + (4 * NumParts)、Y = X + (16 * NumPoints)、Z = Y + 16 + (8 * NumPoints)
2. * 任意指定

ポリゴン Z

ポリゴン Z は、多数のリングで構成されています。リングは、閉じた形で自分自身と交差することのないループになっています。ポリゴン Z には、複数の外部リングを含めることができます。ポリゴン Z のリングのことを、そのポリゴン Z のパーツといいます。

以下がポリゴン Z のフィールドです。

Box Xmin、Ymin、Xmax、Ymax の順序で保管されている、ポリゴン Z の境界ボックス。

NumParts
ポリゴン Z のリングの数。

NumPoints
すべてのリングにあるポイントの合計数。

Parts NumParts の長さの配列。リングごとに、ポイントの配列における最初のポイントの指数を保管します。配列指数は、0 をベースとしています。

Points
NumPoints の長さの配列。ポリゴン Z にある各リングのポイントを端から端まで保管します。リング 1 のポイントの後にリング 2 のポイントが続くというようにつなげます。パーツの配列には、リングごとの最初のポイントの配列指数が保持されます。リング間のポイントの配列には、区切り文字は入りません。

Z Range
Zmin、Zmax の順序で保管される弧の最小と最大の Z 値。

Z Array

NumPoints の長さの配列。ポリゴン Z にある各リングの Z 値を端から端まで保管します。リング 1 の Z 値の後にリング 2 の Z 値が続くようにつなげます。パーツの配列には、リングごとの最初の Z 値の配列指数が保持されます。リング間の Z 値の配列には、区切り文字は入りません。

NumMs

後に続く測定値の数。このフィールドの後に測定値が入力されていない場合、 NumMs は 2 つのゼロ値しかとることができません。測定値が入力されている場合は、NumPoints と等しくなります。

M Range

Mmin、Mmax の順序で保管される、ポリゴン Z の最小と最大の測定値。

M Array

NumPoints の長さの配列。ポリゴン Z にある各リングの測定値を端から端まで保管します。リング 1 の測定値の後にリング 2 の測定値が続くというようにつなげます。パーツの配列には、リングごとの最初の測定値の配列指数が保持されます。リング間の測定値の配列には、区切り文字は入りません。

ポリゴン Z の形状に関する重要な注釈

- リングは閉じています (リングの最初と最後の頂点は必ず同じでなければなりません)。
- ポイントの配列におけるリングの順序は重要ではありません。

表 77. ポリゴン Z のバイト・ストリームの内容

位置	フィールド	値	タイプ	数	順序
バイト 0	形状タイプ	15	整数	1	リトル
バイト 4	Box	Box	倍精度	4	リトル
バイト 36	NumParts	NumParts	整数	1	リトル
バイト 40	NumPoints	NumPoints	整数	1	リトル
バイト 44	Parts	Parts	整数	NumParts	リトル
バイト X	Points	Points	ポイント	NumPoints	リトル
バイト Y	Zmin	Zmin	倍精度	1	リトル
バイト Y+8	Zmax	Zmax	倍精度	1	リトル
バイト Y+16	Zarray	Zarray	倍精度	NumPoints	リトル
バイト Z	NumMs	NumMs	整数	1	リトル

表 77. ポリゴン Z のバイト・ストリームの内容 (続き)

位置	フィールド	値	タイプ	数	順序
バイト Z+4*	Mmin	Mmin	倍精度	1	リトル
バイト Z+12*	Mmax	Mmax	倍精度	1	リトル
バイト Z+20*	Marray	Marray	倍精度	NumPoints	リトル

第3部 付録および後付け

特記事項

本書において、日本では発表されていない IBM 製品 (機械およびプログラム)、プログラミングまたはサービスについて言及または説明する場合があります。しかし、このことは、弊社がこのような IBM 製品、プログラミングまたはサービスを、日本で発表する意図があることを必ずしも示すものではありません。本書でIBM ライセンス・プログラムまたは他の IBM 製品に言及している部分があっても、このことは当該プログラムまたは製品のみが使用可能であることを意味するものではありません。これらのプログラムまたは製品に代えて、 IBM の知的所有権を侵害することのない機能的に同等な他社のプログラム、製品またはサービスを使用することができます。ただし、IBM によって明示的に指定されたものを除き、これらのプログラムまたは製品に関連する稼働の評価および検証はお客様の責任で行っていただきます。

IBM および他社は、本書で説明する主題に関する特許権 (特許出願を含む)、商標権、または著作権を所有している場合があります。本書は、これらの特許権、商標権、および著作権について、本書で明示されている場合を除き、実施権、使用権等を許諾することを意味するものではありません。実施権、使用権等の許諾については、下記の宛先に、書面にてご照会ください。

〒106-0032 東京都港区六本木 3 丁目 2-31
AP 事業所
IBM World Trade Asia Corporation
Intellectual Property Law & Licensing

以下の保証は、国または地域の法律に沿わない場合は、適用されません。 IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

本書に含まれる情報には、技術的に不正確なもの、または誤植が含まれる場合があります。これらに対する変更は、定期的に行われます。これらの変更は、資料の改訂版に含まれます。 IBM は、本書で説明している製品、プログラムに対して、予告なく改良、変更を加える場合があります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するもので

はありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様になんら義務も負わせない適切な方法で、使用もしくは配布することがあります。

本プログラムのライセンス保持者で、 (i) 独自に作成したプログラムとその他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

IBM Canada Limited
Office of the Lab Director
1150 Eglinton Ave. East
North York, Ontario
M3C 1H7
CANADA

本プログラムに関する上記の情報は、適切な条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、 IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、 IBM より提供されます。

本書に含まれるパフォーマンス・データは、制御された環境下で決定されています。したがいまして、その他の稼働環境で得られる結果とは、かなり異なる可能性もあります。一部の測定値は、開発中のシステムを使用している場合があり、これらの測定値が一般的に提供可能なシステムで同様の数値になることを保証するものではありません。さらに、一部の測定値が推定されたものもあります。実測値と異なる場合があります。本書のユーザーは、使用される特定の環境での該当データを確認してください。

IBM 以外の製品については、当該製品の提供者から直接、出版されている資料または一般公開されている情報から入手しました。 IBM は、これらの製品についてはテストを行っておらず、これらの IBM 以外の製品に関する性能、互換性またはその他の主張について確認することはできません。 IBM 以外の製品の機能に対する質問は、それぞれの製品提供者にお問い合わせください。

IBM の将来の方向性または意図については、予告なしに変更または中止する場合があり、 IBM の目的および目標のみを示しているものです。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれていますが、これは説明に具体性を与えるために記載されたものであり、それらの例には、個人、企業、ブランドの、あるいは製品などの名前が含まれている場合があります。それらの名前はすべて架空のものであり、また名称や住所が類似する企業が実在しても、それは偶然に過ぎません。

著作権：

本書に含まれる情報には、サンプル・アプリケーション・プログラムがソース言語の形式で含まれており、様々な、オペレーティング・プラットフォームでのプログラミング技法を示しています。お客様は、これらのサンプル・プログラムが書かれているオペレーティング・プラットフォームでアプリケーション・プログラミング・インターフェースが実行可能となるためのアプリケーション・プログラムを開発、使用、販売または配布もしくは転送する目的のためにのみ、サンプル・プログラムを、IBM に対する別途料金を支払うことなく、複製、変更、配布または転送することができます。これらのサンプルは、すべての条件下で十分にテストを行っていません。したがいまして、IBM は、これらのプログラムの信頼性、実用性または機能について、いかなる保証も負いません。

サンプル・プログラムまたはその改変版の複製物には、全部複製か部分複製かを問わず、次の著作権表示を必ず行うものとします。

商標

次のものは、IBM Corporation の商標です。

ACF/VTAM
AISPO
AIX
AIX/6000
AIXwindows
AnyNet
APPN
AS/400
BookManager
CICS
C Set++
C/370
DATABASE 2
DataHub
DataJoiner
DataPropagator
DataRefresher
DB2
DB2 Connect
DB2 Extenders
DB2 OLAP Server
DB2 Universal Database
Distributed Relational Database Architecture
DRDA
eNetwork
Extended Services
FFST
First Failure Support Technology
IBM
IMS
IMS/ESA
LAN DistanceMVS
MVS/ESA
MVS/XA
Net.Data
OS/2
OS/390
OS/400
PowerPC
QBIC
QMF
RACF
RISC System/6000
RS/6000
S/370
SP
SQL/DS
SQL/400
System/370
System/390
SystemView
VisualAge
VM/ESA
VSE/ESA
VTAM
WebExplorer
WIN-OS/2

次のものは、他社の商標または登録商標です。

Tivoli および NetView は、米国およびその他の国における Tivoli Systems Inc. の商標です。

Microsoft、Windows、Windows NT、および Windows ロゴは Microsoft Corporation の米国およびその他の国における商標です。

Java およびすべての Java 関連の商標およびロゴは Sun Microsystems, Inc. の米国およびその他の国における商標または登録商標です。

UNIX は、The Open Group がライセンスしている米国およびその他の国における登録商標です。

他の会社名、製品名およびサービス名等はそれぞれ各社の商標または登録商標です。

索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

［ア行］

［カ行］

［サ行］

［タ行］

［ナ行］

［ハ行］

［マ行］

［ラ行］

A

B

S

U

W

X

Y

Z

IBM と連絡をとる

技術上の問題がある場合は、時間をとって「問題判別の手引き」に定義されている処置を検討し、それらの提案を実行した後で、DB2 顧客サービスに連絡をとってください。この資料には、DB2 顧客サービスがお客さまを支援するために必要とする情報が説明されています。

製品情報

以下の情報は英語で提供されます。内容は英語版製品に関する情報です。

http://www.ibm.com/software/data/

DB2 World Wide Web ページには、ニュース、製品説明、研修スケジュールなどの DB2 に関する最新情報が提供されています。ただし、提供されている情報は英語です。

http://www.ibm.com/software/data/db2/library/

「DB2 Product and Service Technical Library」では、よくされる質問 (FAQ)、修正内容、資料、および最新の DB2 技術情報などの情報へのアクセスが提供されています。

注: この情報のご提供は英語のみとなりますのでご注意ください。

http://www.elink.ibmlink.ibm.com/pbl/pbl/

「International Publications」注文用 Web サイトでは、マニュアルの注文方法についての情報を提供しています。ただし、提供されている情報は英語です。

http://www.ibm.com/education/certify/

IBM の「Professional Certification Program」Web サイトでは、 DB2 を含むさまざまな IBM 製品の認証テストの情報を提供しています。ただし、提供されている情報は英語です。

ftp.software.ibm.com

匿名でログオンしてください。ディレクトリー `/ps/products/db2` には、 DB2 および多数の他製品に関連したデモ、修正プログラム、情報、およびツールがあります。ただし、提供されている情報は英語です。

comp.databases.ibm-db2, bit.listserv.db2-l

これらのインターネット・ニュースグループは、ユーザーが DB2 製品に関する自分の経験について話し合うために利用できます。ただし、提供されている情報は英語です。

Compuserve: GO IBMDB2

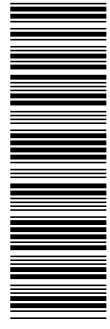
このコマンドを入力すると、IBM DB2 Family forum にアクセスできます。すべての DB2 製品が、このフォーラムでサポートされています。ただし、提供されている情報は英語です。

米国以外の国で IBM に連絡する方法については、 *IBM Software Support Handbook* の Appendix A を参照してください。この資料にアクセスするには、 Web ページ: http://www.ibm.com/support/ にアクセスし、ページの最下部にある「IBM Software Support Handbook」リンク・ボタンを選択します。

注: 国によっては、 IBM が承認している販売業者が、 IBM サポート・センターの代わりにそれら販売業者のサポート・センターに連絡する場合があります。

部品番号: CT7C0JA

Printed in Japan



CT7C0JA

SC88-8624-00

日本アイ・ビー・エム株式会社
〒106-8711　東京都港区六本木3-2-12

Spine information:

IBM® DB2® 地理情報エクステンダー

DB2 地理情報エクステンダー 使用者の手引きおよび解説書

バージョン 7